HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Ringo Lewandrowski

HBOSS -Händlerbund Online-Support System

Konzeption, Planung und prototyptechnische Umsetzung eines Online-Supportsystems

2011

Fakultät Elektro- und Informationstechnik

DIPLOMARBEIT

HBOSS -Händlerbund Online-Support System

Konzeption, Planung und prototyptechnische Umsetzung eines Online-Supportsystems

Autor:
Herr Ringo Lewandrowski

Studiengang:
Multimediatechnik

Seminargruppe:
MK06W1

Erstprüfer:
Prof. Dr. rer. biol. hum. Rudolf Stübner

Zweitprüfer:
Rechtsanwalt Andreas Arlt

Einreichung:
Mittweida, 01.06.2011

Verteidigung/Bewertung:
Mittweida, 2011

Bibliografische Angaben:

Lewandrowski, Ringo: HBOSS - Händlerbund Online-Support-System; Konzeption, Planung und prototyptechnische Umsetzung eines Online-Support-Systems. - 2011 - 77 Seiten,
Mittweida, Hochschule Mittweida (FH), University of Applied Sciences, Fakultät Elektro- und Informationstechnik, Diplomarbeit, 2011

Referat:

Die Diplomarbeit beschäftigt sich mit der Umsetzung eines ticketbasierten Supportsystems, wessen Schwerpunkte in der Netzwerkprogrammierung, der Kommunikation zwischen unterschiedlichen Programmiersprachen und der Nutzung von Datenbank-Schnittstellen liegen. Mittels einer geplanten Anforderungsanalyse werden die Ziele zuerst definiert und im weiteren Verlauf kontinuierlich erörtert.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AG	Aktiengesellschaft
API	Application Programming Interface
ASP	Active Server Pages
C#	C-Sharp
CMS	Content Management System
DOM	Document Object Model
HBOSS	Händlerbund Online Support System
HTTP	Hypertext Transfer Protocol
ID	Identifyer
JDBC	Java Database Connectivity
JDK	Java Developement Kit
JDOM	Java Document Object Model
JNI	Java Native Interface
JSP	Java Server Pages
JVM	Java Virtual Machine
MAC OS	Macintosh Operating System
md5	Message Digest Algorithm 5

MyIsam	My Indexed Sequential Access Method
MySQL	My Structured Query Language
NAT	Network Address Translation
PDA	Personal Digital Assistant
PHP	Hypertext Pre Processor
RDP	Remote Desktop Protocol
SAX	Simple API for XML
SQL	Structured Query Language
TCP	Transport Control Protocol
UDP	User Datagram Protocol
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Einleitung

1.1 Unternehmerische Hintergründe

Die Händlerbund Management AG ist ein junges mittelständiges Unternehmen, dass sich schnell und zielstrebig weiterentwickelt. Der Händlerbund wurde im Oktober 2008 in Leipzig gegründet und er vertritt die Interessen von mehr als 4500 Online-Händlern in den Bereichen der individuellen Rechtstexterstellung bis hin zur Haftungsübernahme. Das Unternehmen macht sich zur Aufgabe die einzelnen Mitglieder untereinander zu vernetzen, deren Lobbyarbeiten zu erledigen und seine Kunden rechtlich abzusichern. Neben dem bereits breit angebotenen rechtlichen Support, möchte der Händlerbund für seine Mitglieder bessere Möglichkeiten bieten um technische Probleme in Bezug auf die Leistungen des Unternehmens zu lösen, diese präventiv zu erörtern und diskret zu bearbeiten. Im Zuge dessen wird das Projekt „Händlerbund Online Support System“ – nachfolgend HBOSS genannt - ins Leben gerufen. HBOSS geht speziell auf die technischen Fragen der Mitglieder ein und bietet somit ein breiteres Spektrum im Bereich des Supports im Unternehmen.

1.2 Zielgruppenbeschreibung

1.2.1 Leser der Diplomarbeit

Die Diplomarbeit beschäftigt sich mit der Umsetzung eines ticketbasierten Supportsystems. Die Schwerpunkte der Arbeit liegen dabei in der Nutzung von Datenbank-Schnittstellen in Java und PHP, deren Kommunikation untereinander und Netzwerkprogrammierung. Im Laufe der nächsten Kapitel werden Methoden betrachtet, wie man sich mit der Programmiersprache Java Zugriff zu einer Datenbank verschafft, zwischen Java und der serverseitigen Scriptsprache PHP kommuniziert oder Client-Server Applikationen entwickelt werden können. Bei dem zuletzt genannten Kriterium handelt es sich speziell um das Übertragen von Daten über das Internet, um beispielsweise eine Remote-Desktop-Verbindung zwischen zwei verschiedenen Systemen zu realisieren.

1.2.2 Anwender der Software

Die Anwender dieses Programms sind technisch nicht versierte oder ältere Menschen, die Probleme darin sehen, die angebotenen Leistungen des Unternehmens sach- und fachgerecht in ihrem Online-Shop einzubinden. Dieses Projekt ist entstanden, um diesen Leuten mit fachlicher Kompetenz und technischem Rat auszuhelfen. Laut interner Statistik beläuft sich die angesprochene Nutzerschicht auf 35 bis 40 Prozent (Michler, 2011).

1.3 Nutzen für das Unternehmen

HBOSS soll dem Unternehmen in erster Linie assistieren. Die Firma erstreckt sich über mehrere Standorte. Somit ist die Kommunikation zwischen den Angestellten meist nur telefonisch regelbar. Wie in den meisten Betrieben existiert auch hier ein Telefondienst, der sämtliche Fragen der Kunden beantwortet. Viele der Fragen beziehen sich aber nicht auf rechtliche oder firmenspezifische Dinge, sondern vielmehr auf technische Probleme. Diese Fragen können nur bedingt beziehungsweise gar nicht beantwortet werden. Wiederum werden die Fragen notiert und telefonisch oder zu Fuß an die technische Abteilung weitergeleitet. Unter anderem, um diesen Mehraufwand zu verringern, soll HBOSS als Prototyp einen „technischen Support“ für die Kunden bieten, indem sie nahezu direkt mit den verantwortlichen Technikern über das Internet, mittels einem personifizierten Programm, kommunizieren können.

2 Fachliches Umfeld

2.1 Java

Java ist eine leistungsstarke und plattformunabhängige Programmiersprache, die erstmals im Jahre 1996 in der Version 1.0, durch die Firma „Sun Microsystems“, erschien. Die aktuelle Version 6.2 und die ab 28.07.2011 geplante Version 7 (Dolphin), zeigen, dass die Programmiersprache Java stets in Entwicklung ist. Auf Grund der Tatsache, dass Java für jeden frei zugänglich ist und ständig Neuentwicklungen im Bereich von Erweiterungen und Rahmenstrukturen (Frameworks) erscheinen, bedient sich die Sprache einer großen Beliebtheit bei mittelständigen Firmen oder Programmierern. Javaprogramme werden nicht, wie es andere Programmiersprachen handhaben, im Maschinencode gespeichert, sondern im Bytecode. Dieser Code lässt sich in einer virtuellen Laufzeitumgebung (JVM – Java Virtual Machine) ausführen. Aus diesem Grund schafft Java den Sprung in die Plattformunabhängigkeit. Denn die virtuelle Umgebung lässt sich nahezu auf allen Rechnerarchitekturen und Betriebssystemen installieren. Um mittels der Sprache Java Applikationen zu erstellen, bieten die Entwickler verschiedene JDK’s (Java Developement Kits) an. Dabei wird ein breites Spektrum an Zielkomponenten abgedeckt, welches von mobilen Endgeräten (Handys oder PDA’s) bis hin zu Serverarchitekturen reicht. Das Entwicklerteam von Java hatte das Ziel eine robuste, objektorientierte und einfache Programmiersprache zu erschaffen, welches ihnen gelungen ist, indem Java im Vergleich zu anderen objektorientierten Sprachen, wie C++ oder C-Sharp, einen reduzierten Sprachumfang besitzt. Da Java Threads[1] beherrscht, können umfangreiche und tiefgreifende Applikationen geschaffen werden.

Die Anwendungsgebiete von Java sind vielseitig. Neben der Entwicklung von Mikroprozessor-Steuerungen oder Sensor-Programmierung, findet die Sprache auch Einsatz im Bereich der Netzwerkprogrammierung, der Erstellung von Client/Server Applikationen, bis hin zur Automaten-Programmierung.

[1] Threads – parallele Programmausführungen

Die Firma „Sun Microsystems“ wurde im Januar 2010 von einem der größten Softwarehersteller, namentlich „Oracle“, aufgekauft. Aber Java wird auch unter diesen Bedingungen weiterhin als Open-Source-Produkt vertrieben.

2.1.1 JDBC und MySQL-Connector

Um mit Java eine Verbindung zu einer bestehenden Datenbank zu schaffen bedient sich der Programmierer an der von „Oracle“ mitgelieferten Klassenbibliothek JDBC (Java Database Connectivity). JDBC ist eine Datenbankschnittstelle, die speziell auf relationale Datenbanksysteme ausgerichtet ist. Die Aufgaben der universellen Datenbankschnittstelle bestehen darin, Datenbankverbindungen aufzubauen und zu verwalten, SQL-Statements an eine Datenbank zu übergeben und deren Ergebnisse rückläufig für Java wieder zur Verfügung zu stellen. Da JDBC speziell für Java entwickelt wurde, benötigt jede spezifische Datenbank einen Treiber, der die Spezifikationen dieser Schnittstelle implementiert. JDBC bietet grundsätzlich vier Typen von Treibern an, um mit Datenbanken zu kommunizieren. Die folgende Tabelle beschreibt die einzelnen Treibertypen und deren Eigenschaften.

Tabelle 1: JDBC Treibertypen

Treiber	Beschreibung
Typ 1 JDBC-ODBC-Bridge	• Zugriff über ODBC-Schnittstelle auf Datenbank • Auf Clients muss ODBC-Treiber installiert sein • Verhältnismäßig langsam, wegen ODBC Zwischenschritt • Nur in einem Intranet empfehlenswert, wo jeweilige ODBC-Treiber auf Clientsystemen installiert sind
Typ 2 Native-API partly Java	• Kommunikation direkt mit Datenbankserver • Auf Clients muss Datenbank-Bibliothek installiert sein • Performanter als Typ 1
Typ 3 JDBC-Net pure Java	• Treiber ist serverbasiert • Wandlung von JDBC Aufrufen in unabhängiges Netzprotokoll • Zugriff auf das bestimmte Datenbanksystem durch eigenes Datenbankmanagementsystem-Protokoll • Zeitintensive Abfragen wegen Server als Middleware • Keine zusätzliche Installation auf Clients notwendig
Typ 4 Native-Protocol pure Java	• Übersetzung der JDBC-Aufrufe in datenbankeigenes Protokoll • Direkter Zugriff auf Datenbank über das Netzwerk • Keine zusätzliche Installation auf Clients notwendig

Die folgende Abbildung 1 verdeutlicht die genannten Eigenschaften der Treibertypen bildlich.

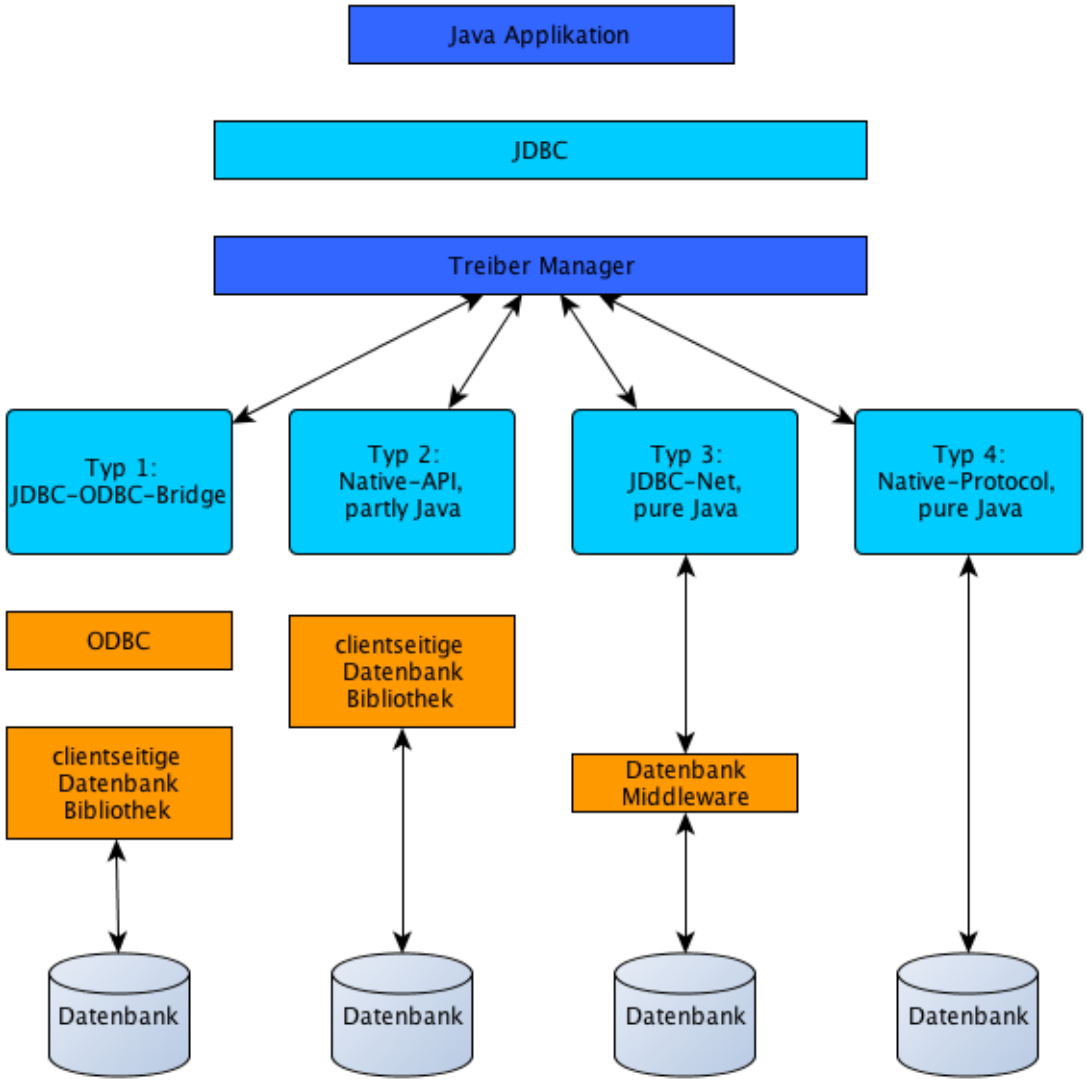

Abbildung 1: JDBC Treibertypen

Die Datenbank MySQL liefert mit dem angebotenen MySQL-Connector für Java einen JDBC-Treiber vom Typ 4. Ist der Treiber in einer Applikation ordnungsgemäß eingebunden, kann diese Anwendung speziell mit MySQL-Datenbanken über das Netzwerk oder Internet kommunizieren, ohne das der Anwender zusätzlich einen Datenbanktreiber installieren muss. Die aktuelle und offizielle Version des JDBC-Treibers MySQL-Connector für Java liegt in der Version 5.1.15 auf der Internetseite von MySQL (http://www.mysql.de) zum Download bereit.

2.1.2 XML, XML-Parser und JDOM

XML (Extensible Markup Language) ist eine Auszeichnungssprache, welche vom World Wide Web Consortium (W3C) im Jahre 1998 standardisiert wurde. Sie dient der hierarchischen Darstellung von Daten in Form von Text. XML ist plattform- und implementationsunabhängig und wird deswegen häufig zum Austausch von Daten zwischen zwei Systemen, vor allem im Internet, verwendet.

Ein Parser ist allgemein bedeutend ein Programm, welches den bestehenden Inhalt eines Dokumentes oder einer Eingabe, in ein für die betreffende Anwendung verwendbares Format übersetzt. Mit SAX (Simple API for XML) und DOM (Document Object Model) stehen der Programmiersprache Java zwei leistungsstarke XML-Parser zur Verfügung. Die beiden Parser unterscheiden sich in ihrer Funktionsweise. Der SAX-Parser arbeitet ereignisorientiert und durchläuft das Dokument sequentiell. Demzufolge, werden dem Parser eine Anzahl von Ereignissen vordefiniert, die während des iterativem „Laufs“ durch das XML-Dokument vorkommen können. Trifft der Parser auf ein solches Ereignis im Dokument, führt er eine Arbeitsroutine durch, auf die der Programmierer speziell reagieren kann. Anders als DOM kann SAX keine Dokumente speichern, aber hat dennoch einen großen Vorteil gegenüber DOM - Den Speicherplatzbedarf im System. Während SAX das Dokument iterativ durchläuft, werden nur die Daten im System gespeichert, die auch wirklich vom Programmierer benötigt werden. Im Gegensatz zu SAX liest DOM das komplette XML-Dokument in den Speicher und kann aus diesem Grund bei großen Dokumenten zu Perfomance-Schwankungen führen. Handelt es sich aber um kleinere Dokumente, besitzt DOM den Vorteil, dass alle Elemente des XML-Dokuments in einem Objektbaum gespeichert werden. Somit kann der Programmier auf die Elemente, sofern er deren Namen kennt, bequem über die Methode „getElementsByTagName(Name_des_Elements)“ zugreifen.

Das Java-Framework JDOM ist keine Abwandlung von dem oben beschriebenen Parser DOM und lehnt demzufolge auch nicht an die Spezifikationen des W3C’s an. JDOM, auch Java Document Object Model genannt, ist eigens für die Programmiersprache Java entwickelt wurden. Es konkurriert in keinerlei Hinsicht mit den bereits vorhanden Parsern.

Vielmehr versucht JDOM, die Vorteile beider Varianten zu vereinen, um es dem Programmierer zu erleichtern, an die erforderlichen Daten zu gelangen. Anders als SAX liest JDOM das Dokument als Baum (ähnlich DOM) in den Hauptspeicher, doch setzt es dafür Java-Klassen ein, anstelle von Nodes (Knoten). Somit ist es dem Programmierer erlaubt, objektorientiert auf die Elemente des Dokumentes zuzugreifen. Durch die Nutzung von spezifischen Java-Klassen, anstelle von typischen DOM-Nodes, steigt die Performance und die erzeugten XML-Objekte können direkt über ihre Eigenschaften und Attributen angesprochen werden. Demzufolge wird der Quellcode übersichtlicher und klarer strukturiert. JDOM hat im weiteren Vergleich mit SAX die Möglichkeit XML-Dokumente zu speichern. Aus diesen Gründen bietet das Java-Document-Object-Model eine echte Alternative zu den Parsern SAX und DOM. Die folgende Abbildung stellt die von JDOM mitgelieferten Klassen zur XML-Behandlung genauer dar.

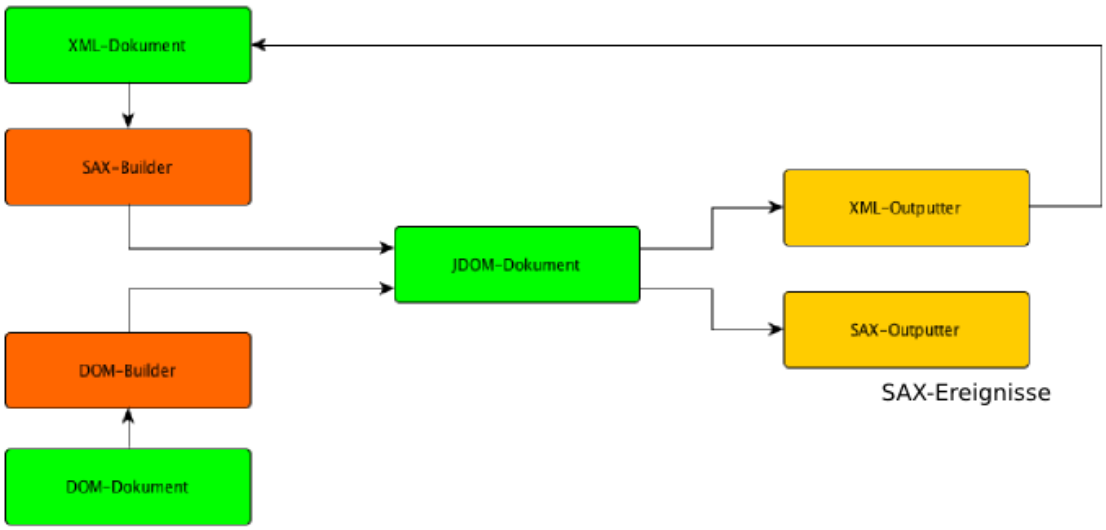

Abbildung 2: JDOM – Struktur

Die oben dargestellte Abbildung 2 zeigt, dass die Klasse XMLOutputter in der Lage ist XML-Dkumente zu erstellen, wobei die zweite Ausgabe-Klasse SAXOutputter in der Lage ist, entsprechend der Struktur des JDOM-Dokuments, SAX-Ereignisse auszulösen. Diese Ereignisse können von anderen Komponenten des Programms interpretiert und somit durch vordefinierte Arbeitsroutinen weiterhin behandelt werden.

2.1.3 JXTA

Der Begriff JXTA leitet sich von dem Namen „juxtapose“ ab und bedeutet übersetzt „Nebeneinanderstellung“. Das Projekt JXTA wurde im Jahr 2001 von Sun Microsystems unter der Führung von Bill Joy und Mike Clary gegründet um eine solide und standardisierte Basis für Peer to Peer (P2P) Kommunikationen zu schaffen. Die JXTA-Protokolle sind unabhängig von Programmiersprache, Betriebssystem und somit dem zugrundeliegenden Transportprotokoll. Diesen Vorteil nutzt JXTA um eine Interoperabilität[2] zwischen zwei gänzlich verschiedenen Systemen zu gewährleisten. Die direkte Kommunikation zwischen zwei Internetnutzern wird heutzutage durch moderne Router und deren Eigenschaften wie NAT (Network Adress Translation) und Firewall erschwert. In einem privaten Netzwerk können Datenpakete auf einem bestimmten Port von einem Rechner gesendet und von einem weiteren empfangen werden. Doch durch die NAT-Technologie funktioniert dies im Internet nicht mehr. Der Router verbirgt die öffentliche IP-Adresse des privaten Netzwerkes und verwaltet die Ports automatisch. Somit werden die Datenpakete im Router auf einen völlig anderen Port nach außen, das heißt ins Internet, geleitet. Versucht man Daten an einen speziellen Teilnehmer im Internet zu senden, der sich hinter einem NAT-System befindet, werden die Pakete vom Ziel-Router ausnahmslos geblockt, da dieser ja nicht „wissen“ kann, an welchen Rechner, in seinem privaten Netzwerk, und an welchen Port das gesendete Paket gerichtet war. Deshalb machen sich JXTA und andere Internetanwendungen wie beispielsweise der VoIP-Vorreiter „Skype“ das Client/Server-Prinzip zu Nutze. Dieses Prinzip wird im Kapitel Implementierung tiefgreifender erörtert.

Ein weiterer Vorteil von JXTA gegenüber normalen Client-Server Anwendungen liegt in der Struktur generell. Das Projekt bietet die Möglichkeit durch sogenannte Peer-Groups ein virtuelles Netzwerk im Internet zu schaffen, wobei jede Peer-Group andere Nutzerrechte besitzen kann (Sams Puplishing, 2002).

[2] Interoperabilität – Zusammenarbeit verschiedener Organisationen, Systemen oder Techniken

2.2 PHP

PHP, auch Hypertext Preprocessor genannt, ist die meistgenutzte Scriptsprache, die zur Erstellung von dynamischen Internetseiten oder Webanwendungen genutzt wird. Die Syntax ist an die Programmiersprache C angelehnt. Dank der freien PHP-Lizenz und der breiten Datenbankunterstützung wird diese Scriptsprache hauptsächlich als serverseitige Programmiersprache verwendet. PHP wird ständig weiterentwickelt und befindet sich seit der Einstellung der Entwicklung an der Version 4 stets im Wandel. Seit der Version 5 bietet PHP verbesserte Möglichkeiten der objektorientierten Programmierung, dass sich durch Kapselung der Daten, Destruktoren und Ausnahmebehandlungen durch sogenannte „Exceptions“ deutlich bemerkbar macht. Die Scriptsprache PHP nutzt eine schwache Typisierung. Das heißt, dass Datentypen von Variablen nicht explizit benannt werden müssen. Dies ist Teil des Konzeptes von PHP und wird auch zukünftig beibehalten. Somit müssen sich Programmierer, die es gewohnt sind mit Programmiersprachen normaler bis starker Typisierung zu arbeiten, auf logische Fehler im Script einstellen. Beispielsweise führt der Vergleich zwischen einer Zeichenkette und einer Zahl zwangsweise nicht zum gewünschten Ergebnis. Ein großer Vorteil dieser Scriptsprache liegt darin, dass sie schnell erlernbar ist. Somit können sich die Programmierer rasch auf die obengenannten Umstände bei der Typisierung der Daten einstellen und diese Fehler durch implizite Typumwandlungen vermeiden.

Die serverseitige Scriptsprache PHP konkurriert hauptsächlich mit zwei anderen Scriptsprachen, die auf einen Webserver ausgeführt werden können – „Active Server Pages“ (ASP) von Microsoft und „Java Server Pages“ von Java beziehungsweise „Oracle“. Weltweit, vor allem in Deutschland, stellen Webserver-Anbieter[3] jedoch mehr Apache-Distributionen als Microsoft-Webserver (IIS) oder javafähige Webserver zur Verfügung. Diese Aussage soll durch die folgende Grafik ergänzend bestätigt werden.

[3] auch Provider genannt; zum Beispiel 1&1 oder All-inkl

Abbildung 3: Marktverteilung von Webserver 2010, (E-Soft Inc., 2010)

Die obige Abbildung 3 zeigt die Marktverteilung von Webservern. Unter normalen Umständen bietet Apache die Scriptsprache PHP an und der Microsoft Internet Information Service die Scriptsprache ASP.

Die Scriptsprache PHP läuft grundsätzlich auf der Apache-Plattform. Doch Apache-Server können auch andere serverbasierte Programmiersprachen geltend machen. Die Scriptsprache JSP (Java-Server-Pages) wird üblicherweise unter dem Apache-Modul „Tomcat“ genutzt. Betrachtet man die Anzahl der installierten Tomcat-Module auf Apache-Webservern, so kristallisiert sich schnell heraus, wie stark verbreitet PHP wirklich ist. (E-Soft Inc., 2010)

2.3 MySQL

MySQL ist ein relationales Datenbanksystem, das als Open-Source-Software angeboten oder als kommerzielle Version vertrieben wird. Der Softwarename setzt sich aus dem Vornamen der Tochter des Mitgründers „My“ und „SQL“ (Structured Query Language) zusammen. Ursprünglich wurde MySQL von dem schwedischen Unternehmen „MySQL AB“ entwickelt, welches im Februar 2008 durch „Sun Microsystems“ aufgekauft wurde.

Seit der Übernahme von „Sun Microsystems“ durch „Oracle“, im Januar 2010, wird der MySQL-Server unter einer Dual-Lizenz (frei oder kommerziell) angeboten. Die Software MySQL ist grundlegend so aufgebaut, dass einem Datenbankmanagement-System mehrere Datenbanken zugeordnet werden können. Jede Datenbank kann mehrere Tabellen von unterschiedlichen Typen (MyISAM oder InnoDB) beinhalten. Diese Typen unterscheiden sich weitgehend in deren Eigenschaften. Während MyISAM eine sehr performante Volltextsuche unterstützt und mehrere Nutzer zeitgleich einen Datensatz lesen können, müssen die Anwender warten bis der letzte Schreibzugriff auf einen Datensatz vollständig abgeschlossen ist um das „eigene“ SQL-Statement auszuführen. Der Typ InnoDB besitzt den Vorteil, dass Statements oder Datenbanktransaktionen abgebrochen werden können. Kommt es zu einem Abbruch einer solchen Transaktion, werden alle Änderungen, die dieser unterlagen, sofort rückgängig gemacht und der betroffene Datensatz ist wieder „frei“ für andere Nutzer oder SQL-Befehlen. Ein weiterer Vorteil dieses Tabellentyps ist die Eigenschaft, Datensätze eineindeutig zu halten. InnoDB prüft hierbei die Relationen und die Korrektheit ihrer Primär- oder Fremdschlüssel zwischen mehreren Tabellen.

3 Anforderungsanalyse

Eine Anforderungsanalyse kann man mit einem sogenannten Pflichtenheft vergleichen. Sie stellt die Funktionalität der Software dar. Diese Analyse beschreibt Beschränkungen und Eigenschaften des Systems, denen der Betrieb der Software und die Entwicklung dieser unterliegen. Für ein auch in der Zukunft wachsendes System wie HBOSS ist es wichtig, korrekt und skalierbar zu planen und dementsprechende Anforderungen an die Software zu stellen.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben auf eine allgemein verständliche Weise, was das Support-Programm HBOSS leisten soll. Sie beschreiben die geforderten Funktionalitäten des Programms und wie es auf Nutzereingaben oder Fehler reagiert.

3.1.1 Registrierung

Der Nutzer kann sich das Programm auf einer speziellen Seite der Internet-Präsenz herunterladen. Dabei werden drei Angaben benötigt: Der Name, die E-Mail-Adresse und sein Passwort. Klickt der Nutzer auf „Download“, so wird per PHP eine XML-Datei mit seinen Daten generiert und anschließend eine ZIP-komprimierte Archiv-Datei erstellt, welche die XML und ausführbare Version des Programms enthält. Folglich wird diese Registrierung auch in der Datenbank vermerkt. Um das Support-Programm als Anwender ausführen zu können, soll das Tool ein automatisiertes Authentifikationsverfahren nutzen. Sind für diesen Nutzer keine persönlichen Einstellungen in der Datenbank hinterlegt, die durch PHP generierte XML-Datei fehlerhaft oder gar manipuliert, so soll das Programm sofort beendet werden und die fehlerhaften Anmeldeversuche in der Datenbank gespeichert werden. In diesem Fall sollen die IP-Adresse und der vollständige PHP-Request registriert werden, um später zu unterscheiden, ob es sich um einen Fehler im System oder einen böswilligen Angriff handelt.

Lässt sich der Nutzer mit seiner eineindeutigen E-Mail-Adresse in der Datenbank auffinden, so sollen ihm die Zugangsdaten für die Datenbank verschlüsselt übertragen werden. Darauf wird im Kapitel 5 – Implementierung erörternd eingegangen.

3.1.2 Ticket auslösen und speichern

HBOSS soll dem Anwender eine Möglichkeit bieten seine technischen Probleme über ein sogenanntes Ticket an die gesamte IT-Abteilung des Betriebs zu leiten. Man kann dieses Verfahren mit dem Verfassen eines Eintrages in ein offizielles Internet-Forum vergleichen. Der einzige Unterschied liegt in der Handhabung. Wenn in einem Forum der Beitrag für alle anderen sichtbar ist, so wird das Ticket äußerst diskret behandelt. Solch ein Ticket beinhaltet zwei nutzerspezifische Angaben. Zum einen den Betreff und zum anderen den Inhalt des niedergeschriebenen Sachverhaltes.

Das Ticket soll mit einer eineindeutigen (folgend auch unique genannten) ID in der Datenbank gespeichert werden. Dabei werden mehrere Tabellen mit verschiedenen Daten gefüllt. Bevor das Ticket in die Datenbank geschrieben wird, sollen mit einem einfachen Algorithmus die Prioritäten geklärt werden. Abbildung 4 zeigt den generellen Ablauf.

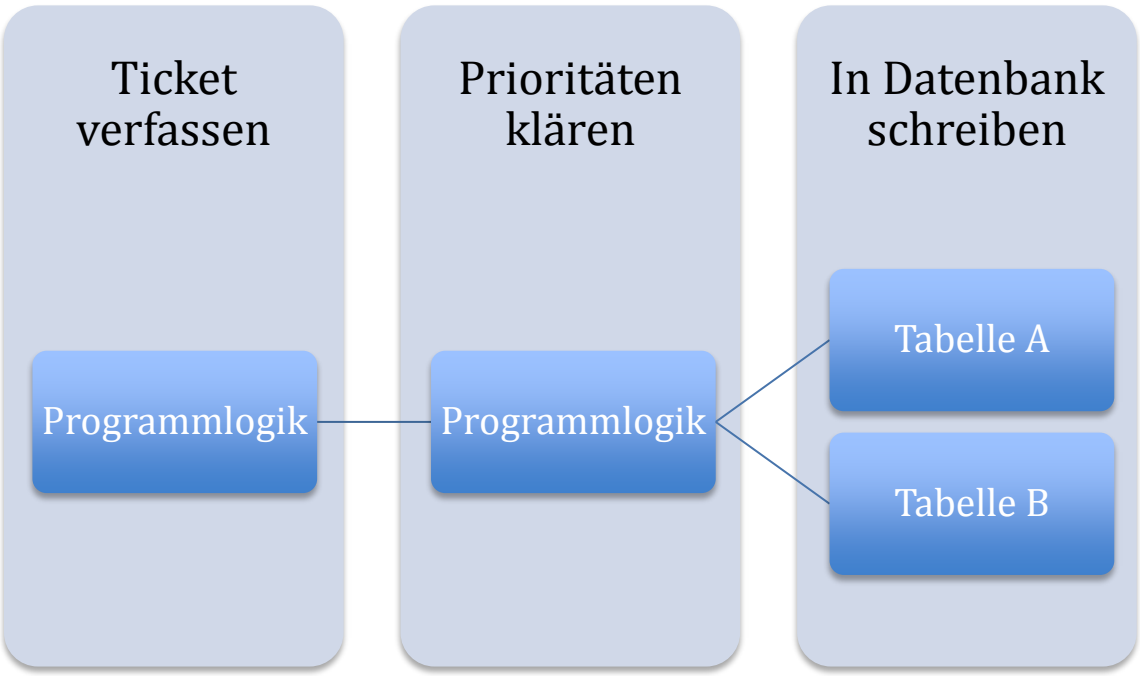

Abbildung 4: Genereller Ablauf Ticketprioritäten

3.1.3 Prioritäten

Ein wesentlicher Bestandteil des Tickets sollen die Prioritäten sein. Unter Rücksprache mit den Juristen im Unternehmen werden gewissen Schlüsselwörtern bestimmte Werte zugeteilt. Der Betreff und der Inhalt des Tickets sollen mittels der Programmlogik auf die Schlüsselwörter untersucht werden. In Abhängigkeit von der Anzahl der gegebenen Schlüsselwörtern in der Datenbank und der gefundenen Wörter im Ticket wird die jeweilige Priorität errechnet. Tabelle 2 und Abbildung 5 zeigen dieses Verfahren beispielhaft anhand fünf gegebener Schlüsselwörter in der Datenbank.

Tabelle 2: Schlüsselwörter mit ihren Prioritäten

Schlüsselwort	Priorität
Problem	1
AGB	3
Rechtstext	3
Logo	1
HTML	1

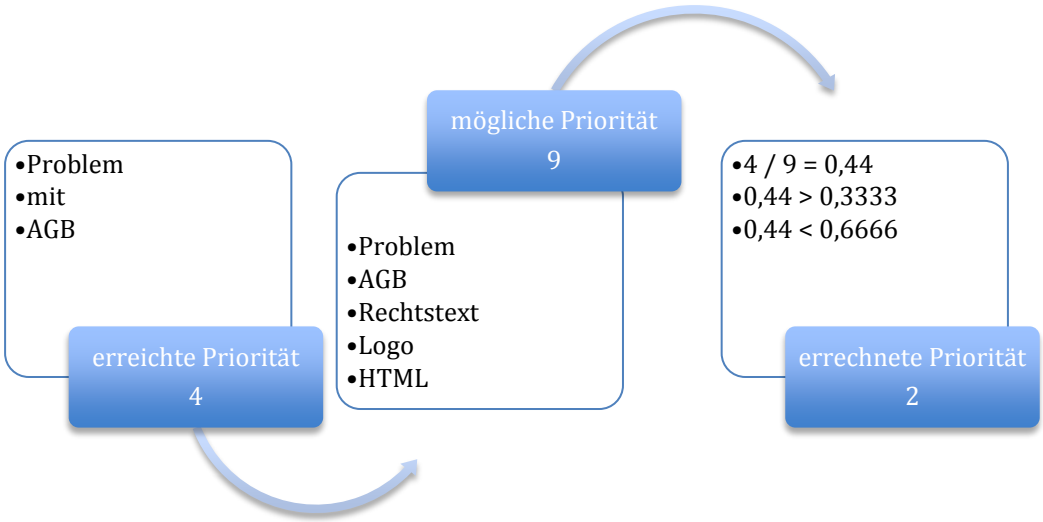

Abbildung 5: Berechnung der Prioritäten im Ticket

Durch die Prozentrechnung wird ein Wert zwischen 0 und 1 berechnet. Innerhalb diesem Intervall liegen drei Stufen $0 < 0{,}33 < 0{,}66 < 1$. Die Prioritäten können daher die Werte „1“, „2“ und „3“ annehmen.

3.1.4 Ticket verarbeiten

HBOSS soll es in zwei verschiedenen Versionen geben. Zum einen das Tool für die Endanwender beziehungsweise die Kunden (Client-Tool) und zum anderen das Tool für die Techniker (Advicer-Tool). Während der Client das Ticket auslöst, reagiert das Advicer-Tool darauf und bearbeitet deren Datensätze in der Datenbank sukzessiv. Je nach Priorität gelangt das ausgelöste Ticket in einer Art Warteschlange und wird an den nächsten freien Techniker weitergeleitet. Unter Beachtung des Arbeitsaufwands jedes einzelnen Advicers soll die Zuordnung der Tickets fair und ausbalanciert stattfinden. Der Advicer hat die Möglichkeit das Ticket direkt zu beantworten, mit dem Anwender per Chat in Kontakt zu treten oder bei einem extrem hohen Prioritätswert und Schwierigkeitsgrad eine Remote-Desktop-Verbindung herzustellen, um eine repräsentative Lösung für das Problem zu finden.

3.1.5 Chat

Nachdem die Mitglieder ein Ticket eingereicht haben, kann der jeweilige Advicer entscheiden, mit welchem Mittel er das technische Problem der Kunden lösen möchte. Um mit dem Mitglied persönlich in Kontakt zu treten soll HBOSS eine Chatfunktion bieten. Es handelt sich hierbei um ein Chat-System, dass nur von den Technikern des Unternehmens gestartet werden kann, um einen unnötigen Missbrauch dieser Funktion zu verhindern. Sowie der Advicer ein Ticket bearbeitet, besteht die Möglichkeit durch einen Klick auf den Namen des Ticketauslösers, mit ihm per Chat in Kontakt zu treten. Auf der Gegenseite öffnet sich parallel ein Dialogfenster, indem das Mitglied den angefragten Chat zulassen oder ablehnen kann.

3.1.6 Remote-Desktop

Ein weiteres Ziel der praktischen Arbeit an dem Projekt HBOSS soll es sein, ein lizenzfreies Modul zu integrieren, welches die Funktionalitäten von Remote-Desktop bietet. Dabei soll es über das Internet möglich sein, den Computer eines Zweiten, mit Hilfe eines Fremdzugriffs, aus der Ferne zu steuern. Darum obliegt der Begriff Remote-Desktop auch der Kategorie Fernwartung. Es gibt bereits viele kostenlose Programme die diese Funktionen anbieten (beispielsweise Teamviewer), doch sind sie nicht mehr gebührenfrei, wenn sie im kommerziellen Bereich, das heißt in einem Betrieb, eingesetzt beziehungsweise verwendet werden. Demzufolge liegt es nahe dieses Verfahren in Form einer Eigenentwicklung zu realisieren. Die Lizenzkosten belaufen sich dadurch auf ein Minimum und die Applikation bleibt für den Programmierer frei erweiterbar.

3.2 Nichtfunktionale Anforderungen

3.2.1 Rahmenbedingungen

Zwischen Idee, Planung und Umsetzung der Diplomarbeit gab es verschiedenste Rahmenbedingungen. Die ursprüngliche Idee basierte auf der Entwicklung eines windowsbasierten Systems, welches es ausdrücklich in der Programmiersprache C-Sharp .Net zu programmieren galt. Unter Verwendung des windowseigenen Protokoll

RDP und der Scriptsprache ASPX sollte eine ActiveX Anwendung geschaffen werden, die es dem Nutzer ermöglicht mit der IT-Abteilung des Händlerbundes direkt oder indirekt zu kommunizieren. Ein wesentlicher Vorteil dieser Programmiersprache besteht darin, dass sämtliche multimediale Hardwarekomponenten über die Schnittstelle DirectX angesprochen werden können. Sollte dies nicht der Fall sein, so besteht dennoch die Möglichkeit die Registrierungsdatei von Windows zu lesen, bearbeiten oder gar eine neue Version, mit angepassten Daten zu generieren. Ein Nachteil dieser Variante lag darin, dass die Mitglieder des Händlerbundes unter Verwendung eines anderen Betriebssystems wie Unix oder MAC OS, von der Nutzung des Programms ausgeschlossen sein würden.

Im Zuge der Reorganisation der Betriebssystemstruktur im Unternehmen von Windows auf MAC OS musste die Idee völlig neu geplant werden. Auf Grund der Umstellung sämtlicher Windows-PCs auf Macintosh-Rechner musste die Programmiersprache plattformunabhängig sein. Die Wahl fiel auf die Programmiersprache Java. Ihr bedeutender Vorteil plattformunabhängig zu sein ist im gleichen Zuge auch der größte Nachteil. Da Java eine virtuelle Laufzeitumgebung für das Ausführen von Bytecode benötigt, leidet die Performance stark darunter. Ein weiterer großer Nachteil von Java lässt sich im Umgang mit multimedialer Hardware erkennen. Im Gegensatz zu Betriebssystemsprachen wie C# oder C++ läuft Java in einem abgeschlossen System. Dieses Sicherheitssystem wird in Java als sogenannte „Sandbox“, zu deutsch Sandkasten, bezeichnet. Die folgende Abbildung stellt diese Sandbox stark vereinfacht dar.

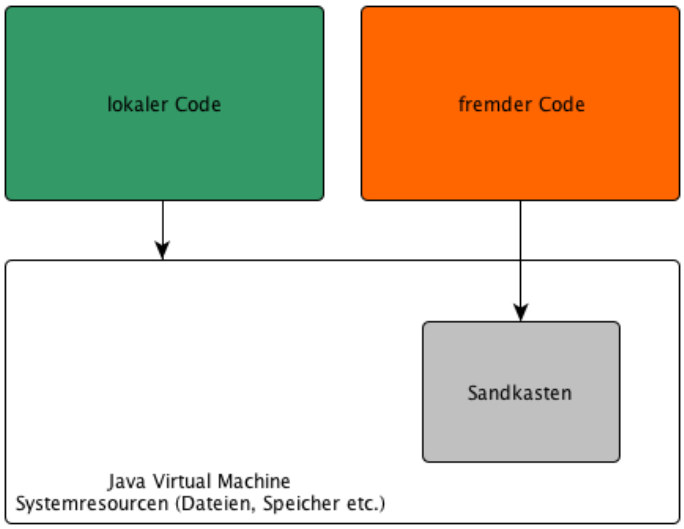

Abbildung 6: Java Sandbox-Model

Das obige Bild zeigt, dass fremder Code keinen direkten Zugriff auf die virtuelle Maschine von Java besitzen darf. Somit ist es mit den üblichen Mitteln Javas nicht möglich direkt auf die Hardware, vor allem Internetkameras, Mikrofone oder Speicher von außen zuzugreifen. Um an die spezifischen betriebssysteminternen Funktionen beziehungsweise Methoden für die Hardware zu gelangen benötigt man die Java-API JNI (Java Native Interface). Unter der Verwendung von JNI ist es mit Java möglich umfangreiche Applikationen für ein gewähltes Betriebssystem zu programmieren, doch verlässt es damit auch die Position der Plattformunabhängigkeit. Auch wenn folgend genannter Nachteil nicht gänzlich auf Java zurückzuführen ist, erschwert er die Arbeit am Projekt HBOSS.

Da das Unternehmen seit seiner Gründung mit speziellen Rahmenverträgen an dessen Internet-Provider gebunden ist und die IT-Infrastrukur im Betrieb keine Möglichkeit bietet einen javabasierten Server an das Internet anzubinden, ist es zwangsläufig notwendig eine fiktive Betrachtung der Umsetzung unter Idealbedingungen von Remote-Desktop und dem Chat vorzunehmen. Nach einer persönlichen Anfrage bei dem bereits erwähnten Provider und einem privaten Anbieter, wurde, unter der Begründung das komplette System und die Verträge ändern zu müssen, verneinend reagiert.

Des Weiteren werden in Kapitel 5 Programme beziehungsweise Quellcodes erörtert, die im Zuge der Nachforschungen zur Realisierung dieser Anforderungen ohne einen solchen Java-Server entstanden.

Während die Arbeit an dem Projekt HBOSS aufgenommen wurde, hat sich im Unternehmen eine weitere äußerst wichtige Ausgangsvariable geändert. Die Verwaltungssoftware für die Mitglieder des Händlerbundes wurde durch eine Anwendung (BATIX) einer Fremdfirma ersetzt. Ursprünglich wurden die Daten der Mitglieder in einer Microsoft-SQL-Datenbank auf einem hausinternen Windowsserver gespeichert. Somit war es den Programmierern schnell und einfach möglich an die erforderlichen Daten zu gelangen.

BATIX ist ein speziell für den Händlerbund zugeschnittenes CMS[4], welches die persönlichen Daten der Mitglieder auf einem externen System speichert. Die Fremdfirma BATIX AG möchte die Zugangsdaten zu deren Datenbank aus Sicherheitsgründen und verständlicherweise zu keiner Kondition preisgeben. Dies erschwert den Zugriff auf die Daten der Mitglieder, die benötigt werden, um eine präzise Authentifizierung oder Softwareverfügbarkeit zu gewährleisten.

3.2.2 Eigenentwicklung oder Standardprogramme

In der Informationstechnik gibt es ein breites Spektrum an Software, die auf bestimmte Probleme mit speziellen Regeln und Funktionen reagieren. Doch diese Programme sind meist nicht auf die benötigten persönlichen Anforderungen zugeschnitten. Um die Anforderungsanalyse ordentlich strukturieren und planen zu können, gilt es die Vor- und Nachteile, die sowohl eine Eigenentwicklung als auch ein Standardprogramm mitbringen abzuwiegen. Gekaufte Programme haben in der Regel einen geschützten Quellcode und dieser kann nur bei Open-Source-Software frei geändert werden. Die Einarbeitung in diesen Quellcode bedarf dann eines sehr hohen Zeitaufwandes.

[4] CMS – Content Management System

Benötigt eine Standardsoftware nutzende Firma ein spezielles Tool oder Plug-In, so ist es stark von dem jeweiligen Fremdunternehmen abhängig, ob dieses zur Verfügung gestellt wird. Eine Eigenentwicklung des Tools oder Plug-Ins spielt dem Unternehmen in so fern in die Hände, dass der Code für die Programmierer frei zugänglich, leichter zu verstehen und somit wesentlich individueller ist. Wartungen und nachträgliche Änderungen sind einfacher zu realisieren als bei vorhandenen Open-Source Projekten. Die Software kann somit mit den sich ändernden Anforderungen wachsen. Ein weiterer Vorteil einer Eigenentwicklung gegenüber einem Standardprogramm ist die sogenannte „Schlankheit“, da sie nur die wirklich benötigten Komponenten beinhaltet. Somit erhöht sich die Übersichtlichkeit des Quellcodes für die Programmierer. Den hohen Lernfaktor und die Aneignung von Wissen und Erfahrung der Programmierer sollte man auch als großen Vorteil verstehen. Auf der anderen Seite lassen sich folgende Nachteile von einer Eigenentwicklung gegenüber einem Standardprogramm aufführen. Die Entwicklungskosten übersteigen zumeist den Beschaffungswert einer kaufbaren Software. Der Zeitaufwand beziehungsweise der Arbeitsaufwand ist in der Regel wesentlich höher, da die Entwicklung einer fehlerfreien Software viel Aufwand erfodert. Open-Source Projekte oder Standardprogramme sind zumeist nahezu fehlerfrei, da größere Fehler bereits entdeckt und durch diverse Updates behoben oder beseitigt wurden.

Im Rahmen dieser Arbeit wird eine Eigenentwicklung bevorzugt, da HBOSS von einem individuellen und schlanken Softwaredesign profitiert und vor allem der Lernfaktor im Vordergrund steht. Auch die Tatsache, dass HBOSS mit seinen aktuellen Modulen und den weiterhin geplanten „Updates“ den Softwaremarkt in noch nicht besiedelte Gebiete betritt, ist erwähnenswert.

4 Datenverarbeitungskonzept

Dieses Kapitel hat die Aufgabe die Strukturen der Programm-Umgebung von HBOSS näher zu erläutern. Im weiteren Verlauf dieses Kapitels werden die verwendeten Schnittstellen, sowie deren Nutzung, auf einem stark abstrahiertem Niveau dargestellt.

4.1 Statische Struktur

Die folgende Abbildung 7 zeigt die statische Umgebungs-Struktur des Support-Programms HBOSS aus einer Art Vogelperspektive. Weiterführend werden die einzelnen verwendeten Komponenten als sogenannte „Blackbox“ dargestellt. Jede Blackbox besitzt Schnittstellen zur Kommunikation mit anderen Komponenten, die mit Hilfe der Grafik bildlich definiert werden können.

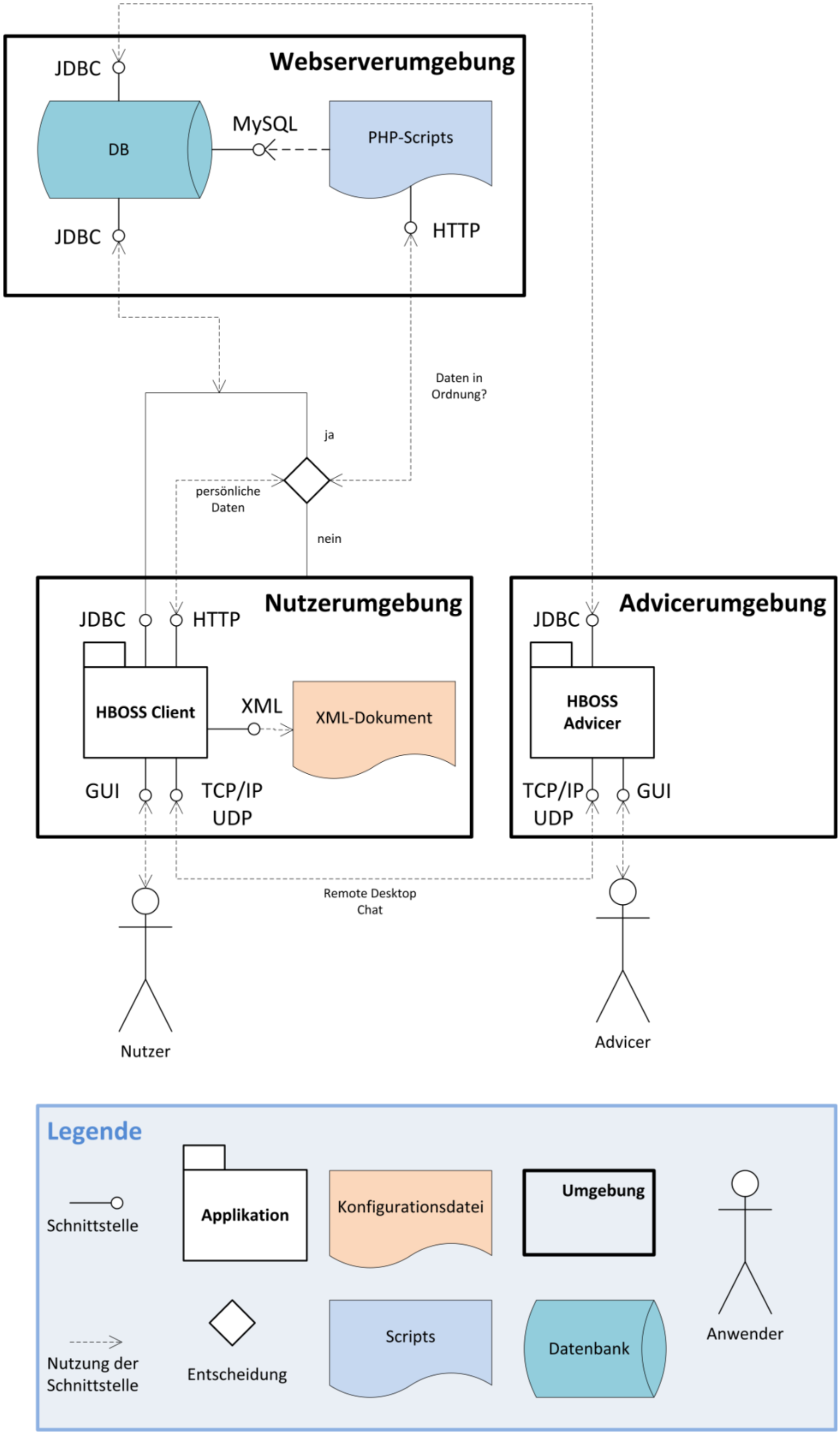

Abbildung 7: DV-Konzept statische Struktur des Ticketsystems

4.2 Datenbankstruktur

Die folgende Grafik zeigt die Datenbankstruktur der Applikation HBOSS. Hierbei werden die Relationen zwischen den einzelnen Tabellen grafisch dargestellt, um die jeweiligen Abhängigkeiten zu verdeutlichen.

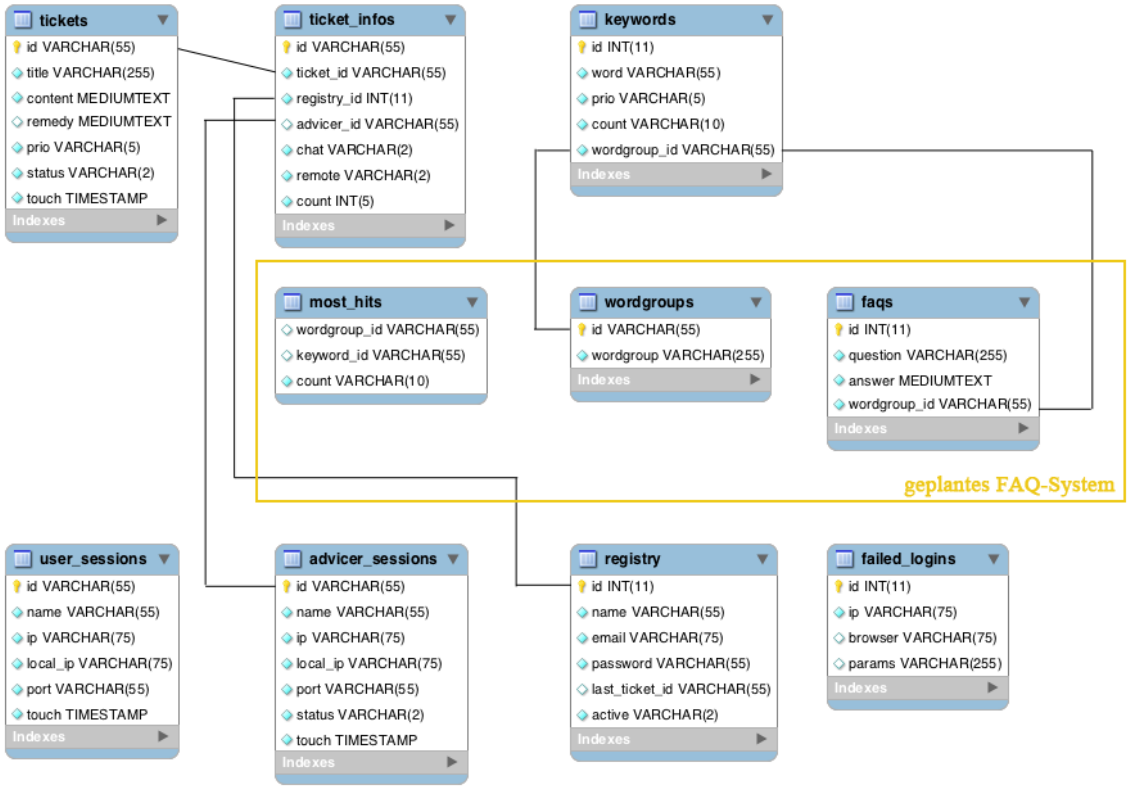

Abbildung 8: Datenbankstruktur

In der obigen Abbildung wird deutlich, dass nicht alle Tabellen in der Relations-Logik beachtet werden. Dies beruht auf der Grundlage, dass die Datenbank von verschiedenen Quellen und aus unterschiedlichen Gründen bearbeitet wird.

5 Implementierung

In diesem Kapitel werden die einzelnen Vorgänge, die zur Implementierung beitragen, fachlich erörtert und wiedergegeben. Die folgenden Quelltext-Auszüge sind Schlüssel-Funktionen und als unterschiedliche Programmier- oder Scriptsprachen gekennzeichnet. Eine weitere Aufgabe dieses Kapitels besteht darin, die Lösungen und Probleme bei der Umsetzung der in Kapitel 3 genannten Anforderungen zu erörtern.

5.1 Dekompilierungsdefizit

Viele Programmierer würden es begrüßen, wenn sie ihre geschriebenen Programme und deren Quelltexte vor Zweiten oder Dritten schützen könnten. Gerade wenn es sich um Closed-Source-Projekte handelt ist diese Problematik ein großes Thema. Es gibt zwar Möglichkeiten seinen Quellcode zu verschlüsseln oder zu verschleiern, doch sind diese Maßnahmen niemals zu 100 Prozent sicher. Eine Vorgehensweise ist zum Beispiel, dass man die Methoden und Variablennamen in der Klasse selbst durch kryptische Zeichen ersetzt. Dieses Verfahren nennt man Obfuskation. Abbildung 9 und Abbildung 10 zeigen dieses Verfahren beispielhaft anhand eines simplen Javacodes (ZDNET, 2007)

```
class OriginalHello {
        public OriginalHello() {
                int number=1;
        }
public String getHello(String helloname){
        return helloname;
}
```

Abbildung 9: Beispiel - Javacode ohne Obfuskation

```
class a {
        public static boolean a;
        public a() {
                int a=1;
        }
public String a(String b){
        return b;
}
```

Abbildung 10: Beispiel - Javacode mit Ofuskation

Das soll das „Lesen“ der Klasse für den Dekompilierer erschweren. Doch selbst die Entwickler[5] solcher Schutzprogramme garantieren keinen zuverlässigen Schutz gegen den obengenannten Code-Diebstahl.

Da dies ein generelles Problem in jeder Programmiersprache darstellt verzichten viele Entwickler ganz und gar auf solche Schutzmaßnahmen, denn der Aufwand den Code zu verschleiern übersteigt den Aufwand den Quelltext mittels moderner Decompiler[6] zu entschlüsseln. Speziell für dieses Projekt wird auch auf solche Schutzmaßnahmen verzichtet.

Da das Programm HBOSS aber Zugang zur Datenbank benötigt und diese Daten nicht direkt im Quellcode der Anwendung hinterlegt werden sollten stützt sich das Tool auf ein eigenes Authentifizierungssystem.

[5] zum Beispiel: http://www.zelix.com/klassmaster/index.html
[6] zum Beispiel: http://en.wikipedia.org/wiki/JAD_(JAva_Decompiler)

5.2 Registrierung

Der Nutzer registriert sich für das Supportprogramm, indem er auf der Internetseite des Unternehmens ein Formular ausfüllt, welches an ein PHP-Script gebunden ist. Das Script arbeitet sukzessiv vier Arbeitsschritte ab. Nach dem klicken auf „senden“ müssen die eingegebenen Daten des Nutzers validiert werden. Das Hauptaugenmerk dieser Validierung obliegt hier der Sicherheit für die bestehende Datenbank des Internetauftritts.

Somit müssen die Nutzereingaben auf bestimmte übergebene Argumente geprüft werden um beispielsweise SQL-Injection-Angriffe zu vermeiden, was im weiteren Verlauf dieses Kapitels näher beschrieben wird.

5.2.1 Registrierungsprozess

Der Anwender füllt auf der firmeneigenen Internetseite ein Formular aus, wie es für eine Registrierung üblich ist, in dem er seinen Namen, sein Passwort und seine E-Mail-Adresse angibt. Das serverseitige PHP-Script, welches für die Validierung dieses Formulars zuständig ist, prüft, ob die E-Mail-Adresse bereits in der Datenbank erfasst wurde. Unter diesem Umstand wird das Script abgebrochen und der Anwender erhält eine Information, dass die übermittelte E-Mail-Adresse bereits vergeben ist beziehungsweise der Anwender bereits registriert ist. Andernfalls wird der Nutzer in der Datenbank aufgenommen. Wie in den Rahmenbedingungen erwähnt, muss dieses PHP-Script zukünftig weiterhin angepasst werden, denn die persönlichen Daten der Mitglieder des Unternehmens, sowie deren ausgewählten Leistungspakte, sind in der Datenbank des Fremdunternehmens „Batix AG“ persistiert. Da sich nur die Mitglieder für das Support-Programm HBOSS registrieren dürfen, welche das entsprechende Leistungspaket gewählt haben, ist eine zukünftige Änderung des folgenden PHP-Quellcodes nicht vermeidbar.

```php
<?php
    if ($_SERVER['REQUEST_METHOD'] == "POST") {
        //db
        require_once 'dbConfig.php';
        //userdata
        $userNameTemp = stripslashes($_POST['userName']);
        $userName = mysql_real_escape_String($userNameTemp);
        $userEmailTemp = stripslashes($_POST['userEmail']);
        $userEmail = mysql_real_escape_String($userEmailTemp);
        $userPasswordTemp = stripslashes($_POST['userPassword']);
        $userPassword = mysql_real_escape_String($userPasswordTemp);

        //check for unique email
        $emailCheck = sprintf("SELECT email from registry
                               WHERE email = '%s'", $userEmail);
        $checkEmail = mysql_query($emailCheck);
        if (mysql_num_rows($checkEmail) > 0) {
            die ("Diese E-Mail-Adresse ist schon vorhanden!");
        }

        //query
        $sql = sprintf("INSERT INTO registry
                        (name, email, password)
                        VALUES
                        ('%s', '%s', '%s')",
        $userName, $userEmail, md5($userPassword));
        $query = mysql_query($sql);
        if ($query) {
            //functions import
            include 'functions.php';
            createXML($userName, $userPassword, $userEmail);
            createZipFile();
            sendMail($userEmail);
            clear();
            echo "Sie haben sich erfolgreich registriert<br/>
            und erhalten in K&uuml;rze eine E-Mail!";
        }
    }
?>
```

Abbildung 11: PHP Code – Registrierung

Die in der Abbildung 11 dargestellte Funktion „stripslashes()“ sowie die Maskierungs-Methode „mysql_real_escape_string()“ werden im Kapitel „SQL-Injection vermeiden“ weitestgehend erläutert. In den Zeilen 16 bis 18 wird eine Abfrage an die Datenbank initialisiert, um zu überprüfen, ob die vom Nutzer eingegebene E-Mail-Adresse bereits in der MySQL-Tabelle „registry“ hinterlegt wurde. Ist dies nicht der Fall, so arbeitet sich das Script sukzessiv zu dem nächsten SQL-Statement vor. In Zeile 22 wird jener SQL-Befehl verfasst, der das Mitglied für das Support-Tool „HBOSS“ registriert und in die Datenbank aufnimmt.

Ein weiterer Schritt des automatisierten Registrierungsprozesses liegt darin, die personifizierte Konfigurationsdatei „options.xml“ zu generieren und diese anschließend in einem zip-komprimierten Archiv, gemeinsam mit dem Programm HBOSS, zu speichern. Die folgenden Abbildungen zeigen die entsprechenden Methoden „createXML()“ und „createZipFile()“, welche jeweils im Anschluss erläutert werden.

```
 3 function createXML($userName, $userPassword, $userEmail) {
 4     $dom = new DOMDocument("1.0");
 5     $root = $dom->createElement("options");
 6
 7     $dom->appendChild($root);
 8
 9     $name= $dom->createElement("name");
10     $root->appendChild($name);
11     $nameContent = $dom->createTextNode($userName);
12     $name->appendChild($nameContent);
13
14     $password = $dom->createElement("password");
15     $root->appendChild($password);
16     $passwordContent = $dom->createTextNode(md5($userPassword));
17     $password->appendChild($passwordContent);
18
19     $email = $dom->createElement("email");
20     $root->appendChild($email);
21     $emailContent = $dom->createTextNode($userEmail);
22     $email->appendChild($emailContent);
23
24     $dom->save("options.xml");
25 }
```

Abbildung 12: PHP Code – createXML()

Die in Abbildung 12 verfasste PHP-Methode „createXML()“ stellt die klassischen DOM-Funktionalitäten des World Wide Web Consortiums dar. DOM-Dokumente werden in nahezu allen Programmiersprachen gleichermaßen bearbeitet. Somit kann man diese Methode als ein demonstrierendes Gegenbeispiel, zu dem vorgestelltem Java-Framework „JDOM“, betrachten. Im Zuge des ersten Schritts, wird in Zeile 4 das XML-DOM-Dokument initialisiert, während in Zeile 5 das Wurzelelement „options“ durch die Funktion „->createElement()“ entsteht. Wie man anhand dieser Abbildung erkennt, ergibt sich für jedes weitere Element eine Art stets wiederkehrender Rhythmus der Syntax. Des Weiteren wird deutlich, wie anspruchsvoll es wäre, eine umfangreichere XML-Datei mittels dem DOM-Objekt zu erzeugen.

```
27 function createZipFile() {
28     $zip = new ZipArchive();
29     if ($zip->open('hboss_client.zip',
30         ZIPARCHIVE::CREATE | ZIPARCHIVE::OVERWRITE) !== TRUE) {
31
32         die ('Leider gab es ein technisches Problem!<br/>
33             Bitte wenden Sie sich an den Support');
34     }
35
36     //list the files to zip
37     $fileList = array('options.xml',
38                       'HBOSSClient.jar',
39                       'lib/jdom.jar',
40                       'lib/mysql-connector-java-5.1.15-bin.jar',
41                       'lib/swing-layout-1.0.4.jar');
42
43     //adding the fileList to zip
44     foreach($fileList as $file) {
45         $zip->addFile($file,$file)
46         or die ('Error adding file '.$file .'to archive!');
47     }
48     $zip->close();
49 }
```

Abbildung 13: PHP Code – createZip()

In Zeile 28 der Abbildung 13 wird das Zip-Archiv-Objekt „$zip“ initialisiert. In den Zeilen 29 bis 34 befindet sich eine formulierte Wenn-Dann-Bedingung, welche sich folgendermaßen erklären lässt. Schlägt die Generierung oder das Überschreiben der Datei namens „hboss_client.zip“ fehl, so wird das PHP-Script durch den Befehl „die()“ abgebrochen. Bei Verwendung der PHP-Funktion „ZipArchiv()“, sollte diese Art von Fehlerbehandlung stets durchgeführt werden, denn jeder Webserver kann eine unterschiedliche Konfiguration beziehungsweise einen anderen Installationsstatus besitzen.

In Zeile 37 wird ein Array mit den Elementen gefüllt, welche dem zu erstellenden Archiv angehören sollen. Dabei handelt es sich um die Programmdateien des Tools HBOSS, sowie die personalisierte Konfigurationsdatei „options.xml“. Die in der Zeile 44 folgende foreach-Schleife iteriert durch das erstellte Array „$fileList“ und fügt dem Zip-Archiv Schritt für Schritt dessen Elemente an. Der Befehl „$zip->close()“ speichert das Prozedere. Um die eben generiete Zip-Datei per E-Mail an den Kunden zu senden bedient sich der Programmierer an der PHP-Funktion namens „mail()“ (siehe Anlagen).

Um Redundanz auf dem Webserver zu vermeiden, müssen die eben erstellten Dateien, nach dem Versenden per E-Mail, wieder entfernt werden. Dies geschieht über die in der Abbildung 11 aufgerufenen Methode „clear()“, welche im folgenden Bild näher betrachtet wird.

```
105 function clear() {
106     //deleting unused files
107     unlink('hboss_client.zip');
108     unlink('options.xml');
109 }
```

Abbildung 14: PHP Code - clear()

Obiges Bild zeigt die Funktion „clear()“, welche mittels der Methode „unlink()“ die nicht mehr benötigten Dateien „hboss_client.zip“ und „options.xml“ von dem Webserver entfernt. Somit ist gewährleistet, dass keine unerwünschten redundanten Daten im System vorhanden sind.

5.2.2 SQL-Injection vermeiden

Den Begriff SQL-Injection, zu deutsch SQL-Einschleusung, kann man als das Ausnutzen von Sicherheitslücken in Bezug auf SQL-Datenbanken definieren. Der Ausführende handelt zu meist mit einem böswilligen Hintergrund und versucht über die jeweilige Anwendung, die den Datenbankzugriff gewährleistet, SQL-Befehle einzuschleusen, um damit die Datenbank auszuspähen, zu manipulieren oder gar das vollständige Serversystem zu kompromittieren. Diese Injektionen sind vor allem dann möglich, wenn Benutzereingaben, ohne eine gründliche Prüfung und Validierung, in den SQL-Befehl der Anwendung gelangen. Solche Nutzereingaben können Sonderzeichen wie den umgekehrten Schrägstrich, das Anführungszeichen und Hochkommata (\,“,’) enthalten, mit denen es möglich ist SQL-Befehle dermaßen zu manipulieren, dass die Statements an die Datenbank mit Hilfe der Sonderzeichen unterbrochen werden können, um im Anschluss logische Bedingungen auf den Wert „TRUE“ zu setzen. Im Zuge dieser Vorgehensweise kann der Angreifer kumulativ die SQL-Befehle in seinem Sinne erweitern und an die benötigten Daten gelangen. Die folgenden Abbildungen in diesem Abschnitt werden das Verfahren der SQL-Injection beispielhaft darlegen.

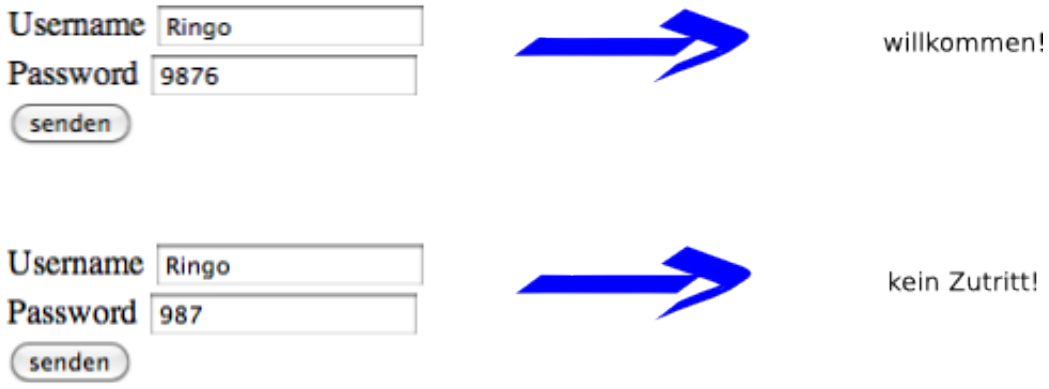

Abbildung 15: HTML-Formular ohne SQL-Einschleusung und die Antwort des PHP-Scripts

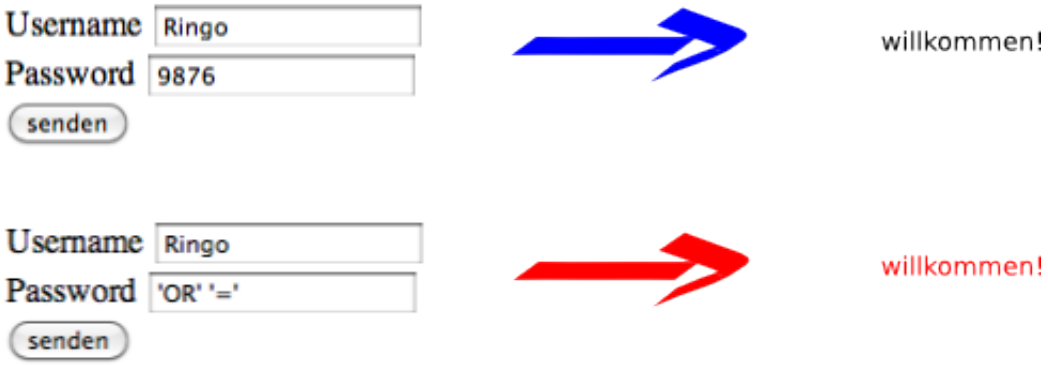

Abbildung 16: HTML-Formular mit SQL-Einschleusung und die Antwort des PHP-Scripts

Folgend wird ein PHP-Script gezeigt, welches für die Validierung des obigen HTML-Formulars zuständig ist.

```php
<?php
include 'dbConfig.php';

//getting the formular-data
$username = $_POST['username'];
$password = $_POST['password'];


//mysql-query
$sql = "SELECT * FROM users WHERE username='$username'
            AND password='$password'";
$query = mysql_query($sql);

if(mysql_num_rows($query)) {
    echo 'willkommen!';
}
else {
    echo 'kein Zutritt!';
}
?>
```

Abbildung 17: SQL-Einschleusung, anfälliges PHP-Script

Abbildung 10 zeigt zwar ein vollkommen richtig programmiertes Script, doch weißt es große Sicherheitslücken auf. In den Zeilen 5 und 6 werden die Benutzereingaben aus dem Formular, ohne vorherige Überprüfung auf gefährliche SQL-Statements und Validierung, übernommen. Somit ist es dem Angreifer möglich, wie in Abbildung 9 gezeigt, ohne Hürden, die in den Zeilen 10 und 11 verfasste SQL-Abfrage zu manipulieren, um in den durch ein Passwort „geschützten“ Bereich zu gelangen. Die Abfrage besteht aus zwei Äquivalenz-Bedingungen. Doch die Manipulation dieses Querys, durch die Eingabe von „ ’ OR ’ ’=’ “ im Passwortfeld des Formulars, führt in der Abfrage dazu, dass aus der Gleichheits-Bedingung für das Passwort eine logische Oder-Bedingung wird. Da „nichts“ immer gleich „nichts“ ist, wird diese neue Oder-Verknüpfung stets wahr und erteilt dem Angreifer in Folge dessen den Zugriff auf den geschützten Bereich.

Um in der Scriptsprache PHP SQL-Injection-Angriffe zu vermeiden kann der Programmierer auf zwei Funktionen der PHP-API zurückgreifen. Im ersten Schritt sollte die Funktion „stripslashes()“ verwendet werden. Diese Methode entfernt alle sogenannten „Slashes“ (/ oder \) aus dem übergebenen Inhalt.

Die wichtigste Methode, namens „mysql_real_escape_string()“, muss immer verwendet werden, um die übermittelten Daten abzusichern, bevor man sie mit einem SQL-Query an die Datenbank übermittelt. Diese Funktion ersetzt sämtliche Befehlszeichen, um einen SQL-Statement zu manipulieren, durch ein „\“. Etwaige Vorgehensweise nennt man Escapen oder auf deutsch Maskieren. Die folgende Abbildung zeigt anhand eines kurzen Beispiels, wie man die PHP-Scripts in Bezug auf SQL-Einschleusung deutlich sicherer machen kann.

```
 1 <?php
 2 include 'dbConfig.php';
 3
 4 //getting the formular-data
 5 $username = stripslashes($_POST['username']);
 6 $password = stripslashes($_POST['password']);
 7
 8
 9 //mysql-query
10 $sql = sprintf("SELECT * FROM users WHERE username='%s'
11              AND password='%s'",
12              mysql_real_escape_string($username),
13              mysql_real_escape_string($password));
14
15 $query = mysql_query($sql);
16
17 if(mysql_num_rows($query)) {
18     echo 'willkommen!';
19 }
20 else {
21     echo 'kein Zutritt!';
22 }
23 ?>
```

Abbildung 18: SQL-Einschleusung, sicheres PHP-Script

Wie bereits erwähnt werden in den Zeilen 5 und 6 (Abbildung 11) die Schrägstriche mit der PHP-Funktion stripslashes() aus den übergebenen Werten entfernt. Eine weitere Verbesserung des Scripts ist die Nutzung der Funktion „sprintf()“. Diese Funktionalität bietet dem Programmierer die Möglichkeit formatierte Zeichenketten zu nutzen. Wie man in den Zeilen 10 bis 13 der obigen Abbildung sehen kann, verwendet sprintf() Argumente und Parameter. Die Argumente geben der Funktion Aufschluss darüber, welcher Typ von Daten erwartet wird und die Parameter liefern, in der Reihenfolge der Argumente, die entsprechenden Daten.

Da die Funktion „mysql_real_escape_string()“ sämtliche SQL-Sonderzeichen entfernt beziehungsweise maskiert, bietet sich dem Angreifer keine Möglichkeit SQL-Injection zu betreiben.

5.3 Authentifizierungssystem

Sobald der Nutzer das Programm startet wird die Konfigurationsdatei „options.xml“ eingelesen. HBOSS sendet diese Daten über einen HTTP-POST-Request an ein PHP-Script auf dem Webserver. Dieses Script prüft, ob die eineindeutige E-Mail-Adresse in der Datenbank vorhanden ist und vergleicht die md5-verschlüsselten Passwörter. Sollten die Daten nicht übereinstimmen bricht das PHP-Script den aktiven Vorgang ab und speichert den kompletten fehlerhaften Request in der Tabelle „failed_logins“. Diese Vorgehensweise lässt später darüber Schlüsse ziehen, ob es sich um einen Fehler in der XML-Datei oder um einen böswilligen Angriff von Dritten handelt. Des Weiteren bietet sie auch eine minimale Sicherheit vor direktem Zugriff auf das Script über die Kommandozeile oder durch einen Webbrowser. Unter der Bedingung, dass die Authentifizierung erfolgreich ist, sendet das PHP-Script die Daten für den Zugriff auf die Datenbank (base64-verschlüsselt) an das Supportsystem HBOSS. Somit ist sichergestellt, dass der Nutzername und das Passwort für die Datenbank nicht direkt im Quellcode der Anwendung stehen. Im Falle einer Dekompilierung sind die Zugangsdaten nicht auffindbar und die Sicherheit der Datenbank ist weiterhin gewährleistet.

Die von dem Webserver-Provider angebotene Nutzerverwaltung der Datenbank ist für eine Anwendung wie HBOSS eher inkompatibel. Dieser bietet pro Datenbank nur einen Nutzer. Somit ist es nicht möglich verschiedene Nutzerrechte zu verteilen und die Sicherheit der Datenbank sinkt stark. Im Idealfall wird eine Datenbank verwendet, in der verschiedene User angelegt werden können. Somit ist es möglich den Endanwendern einen Datenbankzugang zu gewähren, der sich auf das Lesen und Schreiben für ausgewählte Tabellen beschränkt. HBOSS nutzt eine der angebotenen Datenbanken des Providers und nach einer persönlichen Rücksprache mit diesem, sind die übrigen Datenbanken des Unternehmens nicht in Gefahr; Sollte es zu einem nicht erwünschten Fremdzugriff kommen (Höhne, 2011).

5.3.1 Praktische Umsetzung: Authentifizierungssystem

```
41      private void getPersonalData() {
42          try {
43              SAXBuilder saxBuilder = new SAXBuilder();
44              InputSource inputSource = new InputSource(pathToOptionsFile);
45              Document document = saxBuilder.build(inputSource);
46              Element rootElement = document.getRootElement();
47
48              //setting up the data
49              name = rootElement.getChild("name").getText();
50              password = rootElement.getChild("password").getText();
51              email = rootElement.getChild("email").getText();
52          }
53          catch (JDOMException ex) {
54              System.err.println(ex.getMessage());
55          }
56          catch (IOException ex) {
57              System.err.println(ex.getMessage());
58          }
59      }
```

Abbildung 19: Java Code - Einlesen der options.xml

Die obige Abbildung zeigt, wie die Konfigurationsdatei „options.xml“ mit Hilfe des Frameworks JDOM in Java eingelesen wird. Die Datei besteht bekanntlich aus drei Kindelementen. Da jedes wohlgeformte XML Dokument aus einem Mutter-Element (root-element) bestehen muss, ist es mit JDOM sehr einfach auf die einzelnen Elemente des Dokumentes zuzugreifen. Sofern die Namen der Kind-Elemente bekannt sind kann der Programmierer mit Hilfe der von JDOM mitgelieferten Funktion „getChild(name_des_elements).getText()“ auf die hinterlegten Daten eingehen.

Die Abbildung 20 veranschaulicht die Kommunikation zwischen der Anwendung HBOSS und einem auf dem Webserver hinterlegten PHP-Script zur Überprüfung der persönlichen Daten des Anwenders und der Übermittlung der Zugangsdaten für die Datenbank an das Tool HBOSS.

```
61      private void connectToPHP () {
62          try {
63              String urlString = "http://localhost/hboss/solveUser.php";
64              URL url = new URL(urlString);
65              URLConnection connection = url.openConnection();
66              connection.setDoOutput(true);
67              connection.setUseCaches(false);
68
69              String data = "name=" + name +
70                            "&password=" + password +
71                            "&email=" + email;
72
73              OutputStreamWriter writer = new OutputStreamWriter(
74                                          connection.getOutputStream());
75              writer.write(data);
76              writer.flush();
77              writer.close();
78
79              InputStream inputStream = connection.getInputStream();
80              int buffer = 0;
81              result = new StringBuffer();
82              while (buffer >= 0) {
83                  buffer = inputStream.read();
84                  result.append((char) buffer);
85              }
86              if(result.indexOf("null") >= 0) {
87                  JOptionPane.showMessageDialog(null,
88                          "Das System konnte Sie nicht anmelden. "
89                          + "\nBitte wenden Sie sich an den Support.");
90                  System.exit(1);
91              }
92              else if (result.indexOf("falsch") >= 0) {
93                  JOptionPane.showMessageDialog(null,
94                          "Das System konnte Sie nicht anmelden. "
95                          + "\nBitte wenden Sie sich an den Support.");
96                  System.exit(1);
97              }
98              setDatabaseData(new String(result));
99          }
100         catch (Exception ex) {
101             System.err.println(ex.getMessage());
102         }
103     }
```

Abbildung 20: Java Code - Kommunikation mit PHP

Das PHP-Script prüft nach dem eingehenden POST-Request, ob der jeweilige Nutzer anhand der eineindeutigen E-Mail-Adresse in der Datenbank zu finden ist. Stimmen die übermittelten Werte der E-Mail-Adresse und des Passworts aus der XML-Datei mit einem Datensatz in der MySQL-Tabelle „registry“ überein, so werden die Zugangsdaten zur Datenbank vom PHP-Script mit einem Trennzeichen base64-verschlüsselt ausgegeben. Die Abbildung 11 zeigt den Quellcode für das entsprechende Script auf dem Webserver.

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    require_once 'dbConfig.php';

    //check for user in db
    $name = mysql_real_escape_string($_POST['name']);
    $password = mysql_real_escape_string($_POST['password']);
    $email = mysql_real_escape_string($_POST['email']);

    //suppose that "email" is unique
    $query = "SELECT name, password, email FROM registry WHERE email = '$email'";
    $result = mysql_query($query);
    if (mysql_num_rows($result) > 0) {
        $user = mysql_fetch_array($result);
        if (($user['name'] == $name) && ($user['password'] == $password)) {
            //user accepted

            //return databaseData;
            echo base64_encode($server);
            //delimitter
            echo "|";
            echo base64_encode($databaseName);
            //delimitter
            echo "|";
            echo base64_encode($databaseUser);
            //delimitter
            echo "|";
            echo base64_encode($databasePassword);
            //delimitter
            echo "|";
            mysql_close($connection);

        }
        else {
            echo "falsch";
            mysql_close($connection);
        }
    }
    else {
        echo "null";
        mysql_close($connection);
    }
    mysql_close($connection);
}
else {
    echo "Zugriff nicht erlaubt";
}
?>
```

Abbildung 21: PHP Code – Zugangsdaten zur DB veräußern

Um die ausgegebenen verschlüsselten Werte für die Anwendung HBOSS brauchbar zu machen, müssen sie wieder entschlüsselt werden. Dies geschieht über die BASE64-Decoder Klasse von Java. Da die Werte aber über das Trennzeichen „|“ separiert sind müssen sie zuvor noch einzeln in ein String-Array geschrieben werden. Die folgende Abbildung zeigt die Methode „setDatabaseData“ des Support-Programms um das genannte Vorhaben zu realisieren.

```
105      private void setDatabaseData(String data) {
106          BASE64Decoder decoder = new BASE64Decoder();
107          String[] encodedData = data.split("\\|");
108          for (int i=0; i < encodedData.length-1; i++) {
109              try {
110                  HBossClient.dbData[i] = new String(decoder.decodeBuffer(
111                                                  encodedData[i]));
112              }
113              catch (IOException ex) {
114                  System.err.println(ex.getMessage());
115              }
116          }
117      }
```

Abbildung 22: Java Code - Base64 Decoder

Die oben aufgeführte Funktion erwartet die vom PHP-Script übermittelte Zeichenkette als Datentyp String. In der Zeile 107 werden die empfangenen Daten anhand des Trennzeichens „|“ separiert in ein temporäres String-Array gespeichert. Die For-Schleife (ab Zeile 108) decodiert die Daten aus dem temporären Array und speichert sie in ein weiteres, zuvor als public deklariertes, String-Array namens dbData. Somit stehen dem Programm die Zugangsdaten zur Datenbank in Form des gerade gefüllten Arrays dbData während der gesamten Laufzeit zur Verfügung.

Verlief die Kommunikation zwischen der Java-Applikation und dem PHP-Script erfolgreich, wurde der Nutzer in der Datenbank gefunden und stimmen die Passwörter überein, dann stehen dem Tool die Zugangsdaten zur Datenbank ab sofort zur Verfügung – Die Authentifizierung ist hiermit abgeschlossen.

5.4 Ticket einreichen und speichern

Wie in Kapitel 3.1.2 beschrieben können die Nutzer, im Zuge der erfolgreichen Authentifizierung des Programms, ein Ticket mit Betreff und dem dazugehörigen Inhalt verfassen. Dieser Abschnitt befasst sich mit der Thematik, wie die Prioritäten berechnet werden und der systematische Programmablauf für die Ticketerstellung funktioniert. Des Weiteren werden spezielle Funktionen des Java-Quellcodes verständlich in deren Funktionsweise beschrieben.

5.4.1 Schlüsselwörter zur Verfügung stellen

Sobald der Benutzer den Ticket-Dialog über das Kontextmenü „Ticket einreichen“ startet, sendet HBOSS eine MySQL-Abfrage an die Datenbank, die alle verfügbaren Schlüsselwörter, sowie deren jeweilige Prioritäten, als Antwort erhält. Die MySQL-Tabelle „keywords“ stellt diese Daten zur Verfügung und wird regelmäßig von der IT-Abteilung des Unternehmens gewartet, um die Schlüsselwörter zu aktualisieren und deren Prioritätswerte möglichst ausbalanciert zu halten. Um die Verbindung zur Datenbank herzustellen und SQL-Statements an diese zu senden, verwendet das Tool eine eigene Klasse, die vorgefertigte Methoden zum Ausführen bestimmter Abfragen enthält. Nach dem ersten Schritt, welcher die sogenannten Keywords zwischenspeichert, wartet das Java-Programm HBOSS auf die Nutzereingaben des Anwenders beziehungsweise auf das vom Nutzer ausgelöste Event das Ticket versenden zu wollen. Die folgende Abbildung zeigt die Methode „getKeywords()“ der Klasse „MysqlConnection“.

```
63      public CachedRowSet getKeywords() {
64          if (connection != null) {
65              Statement query;
66              try {
67                  query = connection.createStatement();
68                  String sql = "SELECT word, prio FROM keywords";
69
70                  //creating the rowset and exec the query
71                  CachedRowSet rows = new CachedRowSetImpl();
72                  rows.setDataSourceName(databaseName);
73                  rows.setCommand(sql);
74                  rows.execute(connection);
75
76                  //closing the database-connection
77                  connection.close();
78                  instance = null;
79                  return rows;
80              }
81              catch (SQLException ex) {
82                  System.err.println(ex.getMessage());
83                  return null;
84              }
85          }
86          else {
87              return null;
88          }
89      }
```

Abbildung 23: Java Code - MySQL – Keywords

In der Programmiersprache Java gibt es verschiedene Möglichkeiten die Resultate einer SQL-Abfrage zu bearbeiten oder zu verwalten. Zum Einen gibt es das ResultSet, zum Anderen mehrere Arten von RowSets. Diese Strukturen können als Container für Daten definiert werden, welche sich in ihren Eigenschaften unterscheiden. Während das ResultSet nur an die jeweilige Datenbank gebunden ist, können RowSets ihre Daten aus verschiedenen Quellen, wie zum Beispiel einem XML-Dokument, beziehen. RowSets unterscheiden sich aber auch untereinander. Somit sind CachedRowSets nicht von einer permanenten Verbindung zur Datenquelle abhängig, wie normale RowSets. Werden die Daten zur Laufzeit des Programms verändert, können diese dann unter der Bedingung, dass die Verbindung wieder hergestellt wurde, in die Datenbank eingepflegt werden (Ullenboom, 2009). Für den Prototypen HBOSS ist es wichtig, dass keine permanente Verbindung zur Datenbank notwendig ist, um den SQL-Server zu entlasten und dessen Performance zu steigern. Die in der obigen Abbildung gezeigte Funktion liefert den Rückgabewert „CachedRowSet“, in welchem die Daten im Speicher des Systems hinterlegt werden und die Verbindung zur Datenquelle im Anschluss sofort geschlossen wird. In Zeile 68 des obigen Bildes wird die Abfrage verfasst und die folgenden Zeilen generieren das CachedRowSet. Wie man in der Zeile 72 gut erkennen kann, erwartet das erstellte RowSet-Object einen Datenquellennamen. Dies spiegelt die zuvor genannte Eigenschaft wieder, dass RowSets verschiedene Quellen für Daten verwenden können. Das Erstellen und Absenden des SQL-Statements erfolgt durch die Methoden des „rows-Objektes“ setCommand() und execute(). Die Abfrage der Schlüsselwörter ist damit abgeschlossen und das Resultat kann von der Anwendung verwendet werden.

5.4.2 Schlüsselwörter, Prioritäten und das Schreiben in die Datenbank

Wie bereits erwähnt sind die Prioritäten der Schlüsselwörter ausschlaggebend für die weitere Behandlung des Tickets im System. Sobald der Kunde auf „Ticket einsenden“ klickt wird die Zeichenkette des kompletten Betreffs mit den zuvor empfangenen Keywords verglichen. Stimmt mindestens ein Wort der Zeichenkette mit denen aus der Datenbank übermittelten Wörtern überein, so steigt der aktuelle Prioritätswert um den jeweils hinterlegten Wert für das entsprechende Wort. Im nächsten Schritt wird der maximal erreichbare Prioritätswert ermittelt.

Dies geschieht durch eine Addition aller Werte, die im obengenannten CachedRowSet (in diesem Fall keywordRowSet) gespeichert sind. In Folge dieser beiden Arbeitschritte kann nun die erreichte Priorität mittels einer Prozentrechnung berechnet werden. Hierbei wird der aktuelle Prioritätswert mit dem maximal möglichen Wert dividiert und man erhält ein Ergebnis zwischen 0 und 100. Um die Tickets in drei Eskalationstufen einzuteilen, wird zwischen 0 bis 32, 33 bis 65 und 66 bis 100 unterschieden. Die endgültigen Prioritätswerte des Tickets belaufen sich dann auf 1, 2 oder 3. Die folgende Abbildung zeigt die HBOSS-Funktion „checkKeywords()“, welche im Anschluss genauer erörtert wird.

```
126  private void checkKeywords(String title, String content) {
127      String [] titleWords = title.toLowerCase().split("\\s");
128      double priority = 0;
129      double maximumPriority = 0;
130      try {
131          while (keywordRowSet.next()) {
132              for (int i=0; i < titleWords.length; i++) {
133                  if (titleWords[i].contains(keywordRowSet.getString("word"))) {
134                      priority += Integer.parseInt(keywordRowSet.getString("prio"));
135                  }
136              }
137              //possible prio
138              double tempPrio = Integer.parseInt(keywordRowSet.getString("prio"));
139              maximumPriority += tempPrio;
140          }
141          double totalPrio = (double)(priority / maximumPriority * 100);
142          System.out.println(Math.round(totalPrio));
143          if(totalPrio < 33) {
144              database.MysqlConnection.getInstance().setTicket(title, content, 1);
145          }
146          else if (totalPrio >= 33 && totalPrio < 66) {
147              database.MysqlConnection.getInstance().setTicket(title, content, 2);
148          }
149          else if (totalPrio >= 66) {
150              database.MysqlConnection.getInstance().setTicket(title, content, 3);
151          }
152          keywordRowSet.close();
153      }
154      catch (SQLException ex) {
155
156      }
157
158  }
```

Abbildung 24: Java Code – Keywordpriority

In der Zeile 127 wird jedes Wort des Betreffs in ein temporäres String-Array geschrieben. Die Methode „toLowerCase()“ stellt sicher, dass alle Wörter kleingeschrieben werden, um die Bedeutung von Groß- und Kleinschreibung zu minimieren. In den Zeilen 128 und 129 werden die Prioritäten erstmals, mit dem Wert 0, deklariert. Während in Zeile 131 das CachedRowSet mit einem Iterator durchlaufen wird, vergleicht eine For-Schleife die Wörter des Betreffs mit den Wörtern aus dem „keywordRowSet“.

Ein Iterator ist eine Art Datenstrom, der sich sukzessiv von Zeile zu Zeile einer Datenquelle durcharbeitet. In den Zeilen 133 bis 135 wird deutlich gezeigt, wie die aktuelle Priorität addiert wird, wenn es zu einer Übereinstimmung der beiden Wörter kommt. Da sich die Zeilen 138 und 139 noch im etwaigen Iterator befinden, kann an dieser Stelle abschließend nach jeder RowSet-Zeile die maximal mögliche Priorität aufaddiert werden. Die Zeilen 143 bis 151 der obigen Abbildung sind für die Unterscheidung der Eskalationsstufen zuständig. Für jede Stufe wird die Methode „setTicket()“ der Klasse „mysqlConnection“ mit unterschiedlichen Parametern aufgerufen. Das folgende Bild zeigt die eben genannten Methode „setTicket()“, mit welcher das ausgelöste Ticket in der Datenbank gespeichert wird.

```java
public void setTicket(String title, String content, int prio) {
    PreparedStatement preStatement = null;
    if (connection != null) {
        try {
            //ticket
            String ticketID = setID();
            String sql = "INSERT INTO tickets (id, title, content, prio)"
                        + "VALUES (?,?,?,?)";
            preStatement = (PreparedStatement)
                            connection.prepareStatement(sql);
            preStatement.setString(1, ticketID);
            preStatement.setString(2, title);
            preStatement.setString(3, content);
            preStatement.setInt(4, prio);
            preStatement.executeUpdate();
            //ticket info
            String ticketInfoID = setID();
            String sql2 = "INSERT INTO ticket_infos "
                        + "(id, ticket_id, registry_id) "
                        + "VALUES (?,?,(SELECT id FROM registry "
                        + "WHERE email = ?))";
            preStatement = (PreparedStatement)
                            connection.prepareStatement(sql2);
            preStatement.setString(1, ticketInfoID);
            preStatement.setString(2, ticketID);
            String tempData = login.Login.email;
            preStatement.setString(3,tempData);
            preStatement.executeUpdate();
            //wordgroup
            String wordgroupID = setID();
            String sql3 = "INSERT INTO wordgroups (id, wordgroup) "
                        + "VALUES (?,?)";
            preStatement = (PreparedStatement)
                            connection.prepareStatement(sql3);
            preStatement.setString(1, wordgroupID);
            preStatement.setString(2, title);
            preStatement.executeUpdate();
            //registry
            String sql4 = "UPDATE registry SET last_ticket_id = ?"
                        + " WHERE email = ?";
            preStatement = (PreparedStatement)
                            connection.prepareStatement(sql4);
            preStatement.setString(1, ticketID);
            preStatement.setString(2, login.Login.email);
            preStatement.executeUpdate();
            //clear
            preStatement.close();
            JOptionPane.showMessageDialog(null, "erfolgreich erstellt");
            connection.close();
            instance = null;
        }
        catch (SQLException ex) {
            System.err.println(ex);
        }
    }
}
```

Abbildung 25: Java Code – setTicket

Wie bei dem Authentifizierungssystem unter Verwendung von PHP existiert in Java ebenfalls eine Möglichkeit SQL-Injection vermeiden. In der Programmiersprache Java greift man diesbezüglich auch auf eine vorformatierten Zeichenkette zurück und man kann eine bestimmte Äquivalenz zwischen beiden Funktionen erkennen. Der Unterschied beider Varianten liegt lediglich in der Spezialisierung der Funktionen.

Die sprintf-Funktion von PHP kann auf beliebige Zeichenketten im Script angewendet werden, wobei das „prepared Statement“ in Java eine vorbereitete SQL-Anweisung an das Datenbanksystem ist. In einem solchen prepared Statement werden die MySQL-Anweisungen nicht mit den vollständigen Parametern gefüllt, sondern an deren Stelle mit einem „?“ substituiert. Wurde diese vorbereitete Anweisung erfolgreich implementiert, müssen die Fragezeichen folglich zurücksubstituiert werden. In diesem Beispiel-Quellcode heißt das das prepared Statement-Objekt „preStatement“ und besitzt unter anderem die Methode „setString()“. Im Zuge dieser Funktionalität können die ersetzten Parameter der SQL-Abfrage folgendermaßen übergeben werden: setString(1, “Übergabewert“). Die Ziffer „1“ steht in diesem Kontext für die erste vorgenommene Substitution in der Abfrage und der Übergabewert ist der Text, welcher übergeben werden soll. Hierbei ist zu beachten, dass die Methode setString() nur Werte entgegennimmt, die dem Datentyp String unterliegen.

Anhand des obigen Bildes mit dem Quellcode der Methode „setTicket()“ wird beispielhaft gezeigt, wie ein sicheres SQL-Statement an die Datenbank gesendet wird. In den Zeilen 131 bis 134 wird eine umfangreiche beziehungsweise übergreifende Abfrage erzeugt, die in den darauffolgenden zwei Zeilen als prepared Statement definiert werden. Die Zeilen 137, 138 und 140 übergeben dem erzeugten Objekt die Parameter für die SQL-Abfrage. Der nachstehende Befehl „executeUpdate()“ sendet anschließend den geprüften SQL-Befehl an die Datenbank. Des Weiteren ist in Abbildung 25 zu erkennen, dass mehrere SQL-Abfragen hintereinander ausführbar sind. Auf dieser Grundlage werden nahezu zeitgleich vier Tabellen mit Daten gefüllt. Um die Datensätze für die Relationen eineindeutig zu halten, werden die Primärschlüssel der Tabellen durch die Funktion „setID()“ gesetzt. Das folgende Bild zeigt diese einfache Methode und wird im Anschluss erläutert.

```
171      private String setID() {
172          return UUID.randomUUID().toString();
173      }
```

Abbildung 26: Java Code – UUID

Die obige Abbildung enthält den Quellcode zur Generierung einer eineindeutigen Identifikationsnummer. Die Methode „randomUUID()“ liefert eine 128 Bit lange pseudozufällige Zahl. In diesem Fall wird der Begriff „pseudo“ genutzt, weil in der Informationstechnik kein echter Zufalls-Algorithmus existiert. Das heißt, dass der erzeugte Wert mehrmals auftauchen kann, aber die Wahrscheinlichkeit dafür ist sehr gering, da insgesamt 2^{122} Varianten dieser Zahl möglich sind. Deshalb kann dieser Identikator problemlos als Primärschlüssel für Tabellen in einer Datenbank genutzt werden, ohne Gefahr zu laufen, dass dieser bereits von einem anderen Datensatz verwendet wird.

Ist die Funktion „setTicket()“ (Abbildung 25) erfolgreich durchlaufen, wurde das Ticket ausgelöst und befindet sich ab diesem Zeitpunkt in der Datenbank.

5.5 Ticket bearbeiten

Startet ein Mitarbeiter der IT-Abteilung das Advicer-Tool, werden ihm die ungelösten beziehungsweise neu eingetroffenen Tickets in einer Tabelle angezeigt. Die Tickets sind nach deren Prioritäten absteigend sortiert, somit kann der Techniker einen Eintrag aussuchen, bearbeiten und folglich fachgerecht beantworten. Der nächste Abschnitt erläutert die Bearbeitung der Tickets genauer und wird durch Quellcodes unterlegt.

5.5.1 Praktische Umsetzung: Ticket bearbeiten

In diesem Kapitel wird die Thematik behandelt, inwiefern das Ticket durch den Advicer bearbeitet und anschließend in der Datenbank aktualisiert wird. Sobald der Techniker ein Ticket aus der angezeigten Tabelle ausgewählt hat, öffnet sich ein neues Fenster, welches zwei Textfelder und ebenso zwei Checkboxen enthält. Im ersten Textfeld befindet sich der niedergeschriebene Inhalt des Kunden, welcher das Ticket ausgelöst hat. Das zweite Textfeld und die Checkboxen eröffnen dem Techniker die Möglichkeit, um dieses Ticket zu beantworten oder gegebenenfalls einen Chat beziehungsweise eine Remote-Desktop-Verbindung mit dem jeweiligen Benutzer anzufordern. Die folgenden Abbildungen zeigen den Quellcode, um die Tickets aus der Datenbank in die Tabelle zu schreiben, die Tickets zu bearbeiten und folgend die Datensätze in der Datenbank zu aktualisieren.

Im Anschluss eines jeden Bildes folgt eine Erörterung der dargestellten Funktionen.

```
178  private DefaultTableModel unsolvedTickets() {
179      CachedRowSet result = database.MysqlConnection.getInstance()
180                            .getUnsolvedTickets();
181
182      try {
183          DefaultTableModel tableData = new DefaultTableModel();
184          java.sql.ResultSetMetaData resultMeta = result.getMetaData();
185          int columnCount = resultMeta.getColumnCount();
186
187          //toggle first, last row to get count of rows...
188          result.first();
189          result.last();
190          int rowCount = result.getRow();
191          result.beforeFirst();
192
193          Object [][] objectData = new Object[rowCount][columnCount];
194          Object [] columnHeaders = new Object[columnCount];
195
196          //setting up the tableheader
197          for (int i=1; i <= columnCount; i++) {
198              columnHeaders[i-1] = resultMeta.getColumnName(i);
199          }
200
201          //fill tablemodel
202          int currentRow = 0;
203          while (result.next()) {
204              for (int i = 1; i <= columnCount; i++) {
205                  objectData[currentRow][i-1] = result.getString(i);
206              }
207              currentRow++;
208          }
209          tableData.setDataVector(objectData, columnHeaders);
210          return tableData;
211      }
212      catch (SQLException ex) {
213          ex.printStackTrace();
214          return null;
215      }
216
217  }
```

Abbildung 27: Java Code - unsolvedTickets

Der obengezeigte Quellcode stellt die Methode „unsolvedTickets()“ dar, welche einen Rückgabewert vom Typ DefaultTableModel besitzt. Ein solches Tabellen-Modell ist letztendlich ein Daten-Vector, der weitere Vectoren besitzt um die Zellinhalte der Tabelle zu speichern. In der Zeile179 wird durch die Methode „getUnsolvedTickets()“ der Klasse MySQL-Connection ein CachedRowSet namens „result“ mit den ungelösten beziehungsweise neu eingegangenen Tickets gefüllt. Die Funktion „getMetaData()“ liefert Angaben über die Anzahl und Namen der Spalten des Abfrageergebnisses „result“.

Um die Anzahl der Zeilen des Abfrageergebnisses zu erhalten, lässt man den Cursor, wie in den Zeilen 188 bis 190 gezeigt, durch das ResultSet von der ersten bis zur letzten Zeile springen. Die Methode „getRow()“, welche in der letzten Zeile aufgerufen wird, liefert einen Integerwert, der die Anzahl der Zeilen wiedergibt. Mit der Anzahl der Spalten und Zeilen, lässt sich nun die Tabelle mit Hilfe des Tabellen-Modells formen. In Zeile 193 wird ein zweidimensionales Object-Array namens „objectData“ erstellt, welches die Anzahl der Spalten und Namen übergeben bekommt. Die While- und damit verknüpfte For-Schleife in den Zeilen 203 und 204 beginnen iterativ das Abfrageergebnis zu durchlaufen und füllen dabei sukzessiv das Array „objectData“ mit den jeweiligen Daten aus der Zeile „currentRow“ und Spalte i. Sind alle Werte aus den Spalten der aktuellen Zeile in das Array geschrieben worden, springt das Programm aus der For-Schleife und der Iterator veranlasst das Durchlaufen der nächsten Zeile des CachedRowSets. In Zeile 209 bekommt das Tabellen-Modell den endgültigen Daten-Vector, bestehend aus dem gefüllten Array „objectData“ und der jeweiligen Spaltennamen, übergeben. Somit ist die Tabelle geformt und kann als Rückgabewert für folgend aufgeführte Funktion dienen.

```
168      private void setUnsolvedTicketTable() {
169          table = new JTable(unsolvedTickets());
170          scrollPane = new JScrollPane(table);
171          scrollPane.setBounds(10, 50, 600, 400);
172          ticketPanel.add(scrollPane);
173          validate();
174          repaint();
175          tableIsShown = true;
176      }
```

Abbildung 28: Java Code – setUnsolvedTicketTable

In Abbildung 28 wird in Zeile 169 ein Tabellen-Objekt initialisiert, welches auf der Grundlage des oben diskutierten DefaultTableModel basiert. Die Methode „setBounds()“ legt die Position der Tabelle angezeigten Fenster fest. Die weiteren Funktionen wie „validate()“ und „repaint()“ dienen der Aktualisierung der Anzeige in der grafischen Benutzeroberfläche.

Die folgende Abbildung zeigt die Funktion „solveTicket()“, welche die Aufgabe hat, aus der angezeigten Tabelle heraus, einen bestimmten Datensatz zu filtern um ihn im nächsten Schritt an die Klasse „EditTicket“ zu übergeben.

```
260      private void solveTicket() {
261          int selectedRow = table.getSelectedRow();
262          int columnCount = table.getColumnCount();
263          Vector data = new Vector();
264          for (int i=0; i < columnCount; i++) {
265              data.add(table.getValueAt(selectedRow, i));
266          }
267          new EditTicket(data).setVisible(true);
268      }
```

Abbildung 29: Java Code – SolveTicket

In einer grafischen Benutzeroberfläche hat der Anwender die Möglichkeit eine komplette Zeile der Tabelle durch einen Klick zu markieren. Diese Eigenschaft macht sich die in ... dargestellte Methode „solveTicket()“ zu Nutze. In Zeile 261 wird die Zeilennummer in den Speicher geladen, welche ursprünglich angeklickt wurde. Im Zuge der Methoden „getSelectedRow()“ und „getColumnCount()“ kann ein weiterer Daten-Vector, mit Hilfe der in Zeile 264 startenden For-Schleife, mit dem kompletten Datensatz der ausgewählten Zeile gefüllt werden. Der eben erstellte Vector dient dem Konstruktor der Klasse „EditTicket“ als Datengrundlage bei der Initialisierung in Zeile 267 des obigen Quellcodes. Während sich durch diesen Aufruf ein neues Fenster öffnet, werden die Daten des übergebenen Vectors im Konstruktor der Klasse „EditTicket“ genutzt um die entsprechenden Textfelder des neuen Fensters zu füllen. Im oberen Bereich des Fensters befindet sich ein Textfeld mit dem Inhalt des ausgelösten Tickets, während sich im unteren Bereich ein weiteres Textfeld befinden, in dem der Advicer seinen Lösungsvorschlag wiedergeben kann. Des Weiteren befinden sich zwei Checkboxen in diesem Fenster, mit denen es möglich ist, dem Ticket gegebenenfalls den Status von einem benötigten Chat oder gar Remote-Desktop zu übergeben. Der folgende Quellcode ist für das Aktualisieren der Datensätze in der Datenbank zuständig und wird im Anschluss genauer durchleuchtet.

```
if (taRemedy.getText() != "") {
    if (cbChat.isSelected() && !cbRemote.isSelected()) {
        MysqlConnection.getInstance().editTicket(ticketData.get(0).toString(),
                taRemedy.getText(),"1","0");
    }
    else if(cbRemote.isSelected() && !cbChat.isSelected()) {
        MysqlConnection.getInstance().editTicket(ticketData.get(0).toString(),
                taRemedy.getText(),"0","1");
    }
    else if(cbChat.isSelected() && cbRemote.isSelected()) {
        MysqlConnection.getInstance().editTicket(ticketData.get(0).toString(),
                taRemedy.getText(),"1","1");
    }
    else {
        MysqlConnection.getInstance().editTicket(ticketData.get(0).toString(),
                taRemedy.getText(),"0","0");
    }

}
```

Abbildung 30: Java Code – EditTicket

Der Quellcode der Abbildung 30 weißt eine Vielzahl von If-Bedingungen auf, um zu unterscheiden, welche Checkboxes angeklickt wurden und welches SQL-Statement an die Datenbank geschickt wird. Die Variable „ticketData“ ist ein Vector, welcher die Daten aus der Tabelle beinhaltet. Aus diesem Grund ist es möglich mit der Funktion „ticketData.get(0)“ auf den ersten Eintrag, in diesem Fall auf die eineindeutige Ticket-ID des Datensatzes, des Vectors zuzugreifen. Im Zuge dessen lässt sich das bearbeitete Ticket durch den Primärschlüssel ohne großen Aufwand in der Datenbank aktualisieren.

5.6 Remote-Desktop

Dieses Kapitel beschäftigt sich mit der Planung und Umsetzung einer Remote-Desktop-Verbindung zwischen zwei Computern über das Internet. Heutzutage ist es üblich, dass Firmen oder private Haushalte ihren Internetzugang durch einen Router, mit eingebautem oder zuvor geschaltetem Modem, in Anspruch nehmen. Ein Router hat die Eigenschaft aber auch Aufgabe, wegen der zunehmenden Knappheit an öffentlichen IP-Adressen (IPv4[7]), private Netzwerke von der Öffentlichkeit abzugrenzen. Dies geschieht über das Verfahren NAT (Network Address Translation). Der nächste Abschnitt erläutert die Funktionsweise von NAT.

[7] IPv4 – 32Bit-Adresse

5.6.1 Network Address Translation

Im Zuge der zunehmenden Knappheit an öffentlichen Internetadressen im IPv4 (bald durch IPv6[8] ersetzt) Adressbereich und der Wunsch der Gesellschaft, private Netzwerke mit dem Internet zu verbinden, wurde NAT eingeführt. NAT macht es sich zur Aufgabe private Netzwerke von dem öffentlichen Netzwerk Internet abzugrenzen. Während das Gateway des Routers die öffentliche Internetadresse zur Kommunikation mit dem Internet weiß, wissen die Teilnehmer des privaten Netzwerkes nur die Adresse zu diesem Gateway. So ist es gewährleistet, dass die einzelnen Rechner über das Gateway mit dem Internet kommunizieren können, aber dabei keine eigene öffentliche IP-Adresse benötigen. Die folgende Abbildung soll dies veranschaulichen.

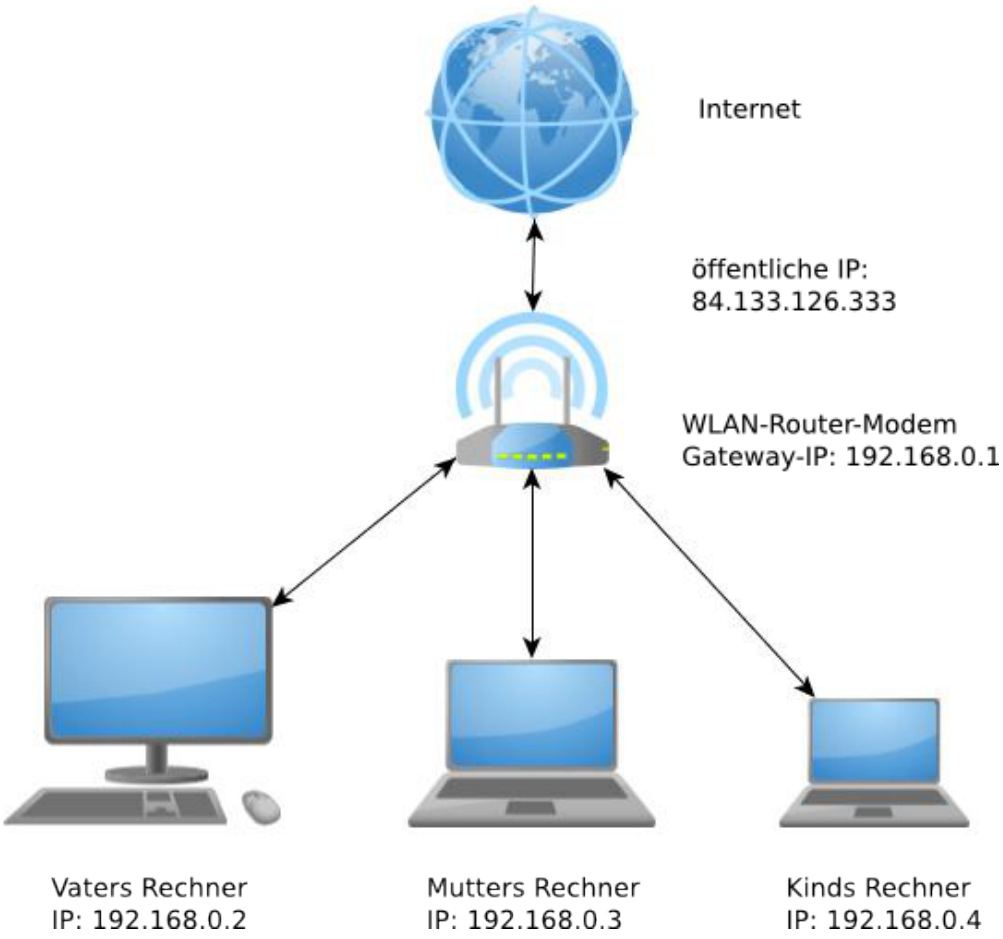

Abbildung 31: NAT

[8] IPv6 – 128Bit-Adresse

Network Address Translation verfügt über zwei verschiedene Herangehensweisen: Source-NAT und Destination-NAT. Im Laufe dieses Abschnitts wird die Variante genauer beschrieben, welche am häufigsten bei kleinen bis mittelständigen Unternehmen oder in privaten Haushalten zum Einsatz kommen, Source-NAT. Das Vorgehen von diesem NAT-Typen kann man folgendermaßen, unter Verwendung obiger Abbildung, beschreiben. Sobald Vaters Rechner aus dem privaten Netzwerk versucht mit einem Rechner oder System aus einem anderen Netz zu kommunizieren, sendet er seine Datenpakete und die Zieladresse an das netzwerkinterne Gateway. Danach wird seine lokale IP-Adresse mit der öffentlichen Internet Protokoll-Adresse durch den Router ersetzt.

Tabelle 3: Source-NAT - Quell- und Ziel-IP's

Quell-IP:Port	Ziel-IP:Port	Quell-IP:Port nach NAT	Ziel-IP:Port nach NAT
192.168.0.2:9999	763.555.789.6	84.133.126.333:80	763.555.789.6:80
192.168.0.2:80	763.555.789.6	84.133.126.333:6464	763.555.789.6:6464

Wie man in der gezeigten Tabelle erkennen kann, ändert der Router nicht nur die Quell-Adressen, sondern auch die Ports auf denen die Datenpakete versendet werden. Dies hat zur Folge, dass der sendende Anwender niemals genau weiß, welche öffentliche Adresse er nutzt und auf welchem Port er seine Datenpakete sendet. Des Weiteren weiß auch die Gegenstelle nicht, von welchem Rechner aus dem privaten Netzwerk das empfangene Datenpaket kam und welcher Port ursprünglich verwendet wurde. Dieses Problem könnte man mittels „Port-Forwarding“ (Portweiterleitung) an den jeweiligen Routern zwar umgehen, doch ist es eine wage Zumutung, von technisch nicht versierten Menschen (Anwender der Software) zu erwarten, dass sie ihren Router selbst so konfigurieren, dass die gesendeten und empfangenen Datenpakete auf definierten Ports nicht umgeleitet werden. Aus diesem Grund werden in dieser Arbeit, wie in den Rahmenbedingungen bereits erwähnt, zwei verschiedene Arten verfolgt, um eine Remote-Desktop Applikation zu realisieren. Zum ersten die fiktive Betrachtung unter idealen Vorraussetzungen und zum zweiten die, im Zuge der Forschung zur Umsetzung dieser Funktion ohne Idealbedingungen, entstanden Ansätze.

5.6.2 NAT-Traversal und fiktive Betrachtung von Remote-Desktop

Eine Remote-Desktop-Applikation, oder auch andere Programme, die eine direkte Endpunkt zu Endpunkt Verbindung über das Internet benötigen, nutzen im Idealfall einen Server mit einer festen öffentlichen Internetadresse oder einer festen Domain, die immer auf die aktuelle Adresse des Servers verweist. In diesem Fall wird ein Server verwendet, auf dem die Java Virtual Machine installiert ist und damit javabasierte Serveranwendungen ausführbar sind. Die folgende Abbildung stellt eine übliche Server/Client Umgebung dar, um eine direkte Verbindung zwischen zwei Endpunkten zu gewährleisten.

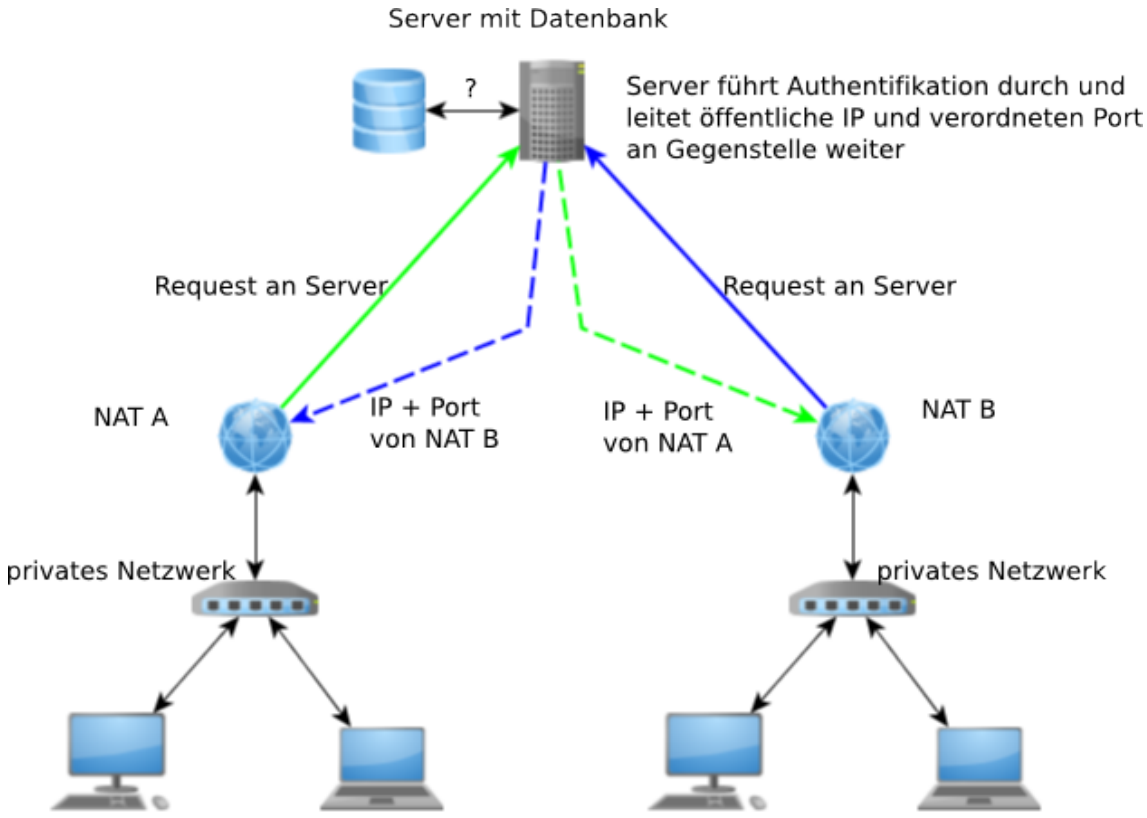

Abbildung 32: NAT-Durchdringung via Server

Die obige Abbildung 32 stellt eine ideale Umgebung für direkte Kommunikation zwischen zwei oder mehreren Endpunkten dar. In der Datenbank befindet sich eine Tabelle, welche die persönlichen Daten (Name, E-Mail und Passwort) der möglichen Benutzer verwaltet. Der Server besitzt eine feste IP-Adresse und hat lediglich zwei Aufgaben.

Die erste Aufgabe besteht darin, die Nutzer zu authentifizieren und anschließend deren jeweilige, durch die NAT preisgegebene, öffentliche IP-Adresse und dem, vom jeweiligen Router zugeordneten Port, im betreffenden Datensatz der Datenbank zu aktualisieren. Die letzte Aufgabe des Servers liegt in der Verwaltung der Anfragen. Möchte beispielsweise NAT A mit NAT B kommunizieren, so ist der Server dafür zuständig, ob B überhaupt erreichbar ist und das die IP-Adresse zuzüglich dem verwendeten Port von B an A weitergeleitet wird. In Folge dessen kann die Verbindung zwischen den einzelnen Endpunkten hergestellt werden und der Datenaustausch beginnen.

Java bietet mit Hilfe der „net“ Bibliothek drei Klassen, die zur Realisierung von Netzwerk-Applikationen dienen - ServerSocket, Socket und DatagramSocket. Während sich ServerSocket und Socket auf das verbindungsorientierte Netzwerk-Protokoll TCP stützen, obliegt das DatagramSocket dem verbindungslosen Protokoll UDP. Der wesentliche Unterschied beider Protokolle liegt darin, dass verbindungsorientierte Datenpakete beim Empfänger durch eine Serialisierung inklusive Prüfsummen vollständig und sortiert ankommen, wobei hingegen verbindungslose Datenpakete willkürlich ohne eine Überprüfung, ob die letzten übermittelten Daten angekommen sind, einfach weiter übertragen werden. Erschwerend für den Empfänger eines solchen Paketes kommt hinzu, dass die einzelnen Teildaten dieses Paketes jeweils anders durch das Internet transportiert werden können. In einem solchen Fall kann beispielsweise der erste Teil des UDP-Paketes an dritter Stelle aller empfangenen Daten ankommen. Somit ist das komplett empfangene Datenpaket völlig unbrauchbar. Aus diesem Grund wird UDP nicht in Programmen verwendet, in denen es auf eine sichere Datenübertragung ankommt, sondern vielmehr bei Anwendungen, die mehr Wert auf Quantität legen als auf Qualität, wie zum Beispiel Video-Live-Übertragungen oder Videokonferenzen.

Da die einzelnen Anwender die IP-Endpunkte und Ports der Gegenstelle wissen (Abbildung 32), können die Socket-Klassen von Java verwendet werden um die direkte Verbindung herzustellen. Für eine Remote-Desktop Anwendung ist es wichtig zwischen Bildschirmsender und Bildschirmempfänger zu unterscheiden, denn der Sender nimmt zwar die Rolle des Servers ein, aber der Empfänger ist derjenige, der letztlich die Kontrolle über dieses System verfügt.

Um ein Bild des Bildschirms an die Gegenstelle zu übertragen muss zuvor ein sogenannter Screenshot (Bildschirmfoto) erstellt werden. Die Java-Klasse „Robot“ bietet mit der Methode „createScreenCapture(Dimensionsrechteck)“ die Möglichkeit genau diese Funktion zu realisieren. Sie liefert als Resultat ein nicht komprimiertes Bild der Größe, welche im Dimensionsrechteck definiert wurde. Um das Netzwerk nicht zu überlasten, sollte aus diesem Grund das erstellte Bild vor der Übertragung in ein komprimiertes Bildformat encodiert werden. Mit Hilfe der Java-Klasse JPEGEncoder ist es möglich dieses Vorhaben zu realisieren und wird im nächsten Teilkapitel anhand exemplarischer Quellcodes in ihrer Funktionsweise näher durchleuchtet.

5.6.2.1 Exemplarische Funktions-Quellcodes - Remote-Desktop (fiktiv)

Dieses Teilkapitel stellt exemplarisch Funktionen von Java vor, welche zur Umsetzung einer Remote-Desktop-Server Applikation dienen könnten. Dieser Abschnitt behandelt eine threadfähige Java-Klasse, die aus einem Konstruktor, einer Main-Methode und einer partiell erstellten Thread-Methode besteht, welche sukzessiv genauer erörtert werden. Der folgende Quellcode zeigt den Konstruktor der eben genannten Klasse „RemoteDesktopServer“.

```
33  public RemoteDesktopServer() {
34      try {
35          //create robot object
36          this.robot = new Robot();
37
38          //get the display-width and height
39          //to prepare screenshots
40          DisplayMode displayMode = GraphicsEnvironment
41                          .getLocalGraphicsEnvironment()
42                          .getDefaultScreenDevice()
43                          .getDisplayMode();
44
45          //setting up the dsiplay dimension
46          this.screen = new Rectangle(0, 0,
47                      displayMode.getWidth(),
48                      displayMode.getHeight());
49
50          //setting the servercomitted clientport
51          this.clientPort = getClientPort();
52      }
53      catch (AWTException ex) {
54          System.err.println(ex.getMessage());
55      }
56
57  }
```

Abbildung 33: fiktiv Remote-Desktop - Konstruktor

Im oben gezeigten Quellcode wird in Zeile 36 das Java-Objekt Robot initialisiert, mit dem es später möglich ist Bildschirmfotos zu produzieren. Die Klasse DisplayMode greift ein wenig tiefer in das System ein und beschafft mit deren Methoden die Informationen zu den Bildschirmen, welche die Anwender benutzen. Nutzt ein Anwender beispielsweise zwei oder gar mehrere Monitore, so wird mittels der Funktion „getDefaultScreenDevice()“ stets der Hauptmonitor und dessen Eigenschaften ausgewählt. In Zeile 46 wird der, zuvor als „Rectancle“ deklarierten, Variable „screen“ eine neue Dimension veräußert, welche abhängig von der Auflösung des Hauptmonitors ist. Die Zeile 51 stellt einen Verweis auf die „getClientPort()“-Methode dar. Diese wird, wie in folgender Abbildung 34 dargestellt, implemtiert.

```
82  private int getClientPort() {
83      clientPort = Integer.parseInt(clientData[1]);
84      return clientPort;
85  }
```

Abbildung 34: fiktiv Remote-Desktop – getClientPort()

Um die in Abbildung 34 gezeigte Funktionalität zu erörtern, müssen noch einige Fakten geklärt werden. Das Array „clientData“ wurde zuvor mit den gelieferten Daten von dem Java-Server, mit einer ähnlichen Methode, wie in dem Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben, gefüllt. Der Klasse „RemoteDesktopServer“ stehen auf Grund dessen die Konnektivitätsbedingungen zur jeweiligen Gegenstelle zur Verfügung.

Die Methode „accept()“ der Klasse ServerSocket besitzt die Eigenschaft, keine weiteren Befehlsfolgen auszuführen, bis ein Client die Verbindung hergestellt hat. Dies kann bei einer schlecht durchdachten Programmlogik dazu führen, dass das erstellte Programm „einfriert“, solange diese Methode keinen Teilnehmer oder Abschluss findet. Aus diesem Grund wird in diesem Beispiel eine partiell erstellte Thread-Methode verwendet, um diese Klasse zu implementieren. Die folgende Abbildung 35 veranschaulicht das Verhalten der Methode „manageConnection()“ beispielhaft.

```
63  private Runnable manageConnection() {
64      return new Runnable() {
65
66          public void run() {
67              try {
68                  ServerSocket serverSocket =
69                          new ServerSocket(clientPort);
70
71                  while (true) {
72                      clientSocket = serverSocket.accept();
73                  }
74              }
75              catch (IOException ex) {
76                  System.err.println(ex.getMessage());
77              }
78          }
79      };
80  }
```

Abbildung 35: fiktiv Remote-Desktop: manageConnection()

Wie man anhand der obigen Abbildung erkennen kann, liefert diese Methode einen Rückgabewert des Typs „Runnable“. Dabei handelt es sich um ein Java-Interface, welches die Eigenschaften von der Klasse Thread erbt und somit auch threadfähig ist. Aus diesem Grund ist es möglich diese Methode nebenläufig zu dem eigentlichen Programm-Code auszuführen, ohne die Applikation „festzufahren“.

Um den Remote-Desktop-Server endgültig zu starten wird eine weitere Methode benötigt, welche regelmäßig die Screenshots erstellt, die erstellten Bilder encodiert und folglich als ersten Schritt die Methode „manageConnection()“ ausführt. Der folgende Quellcode behandelt die Funktion „run()“ der Klasse RemoteDesktopServer und wird anschließend genauer durchleuchtet.

```
94  public void run() {
95      manageConnection();
96      while (true) {
97          //taking screenshot
98          BufferedImage screenImage = takeScreenShot();
99          ByteArrayOutputStream byteOut =
100                 new ByteArrayOutputStream();
101
102         //encode the image for better performance
103         JPEGImageEncoder encoder =
104                 JPEGCodec.createJPEGEncoder(byteOut);
105         try {
106             encoder.encode(screenImage);
107         }
108         catch (Exception ex) {
109             System.err.println(ex.getMessage());
110         }
111         try {
112             //sending data to client
113             clientSocket.getOutputStream()
114                         .write(byteOut.toByteArray());
115         }
116         catch (IOException ex) {
117             System.err.println(ex.getMessage());
118         }
119         try {
120             TimeUnit.MILLISECONDS.sleep(250);
121         }
122         catch (InterruptedException ex) {
123             System.err.println(ex.getMessage());
124         }
125     }
}
```

Abbildung 36: fiktiv Remote-Desktop - run()

Die in Zeile 95 erstellte While-Schleife in der obigen Abbildung stellt sicher, dass die eingeschlossenen Anweisungen ausgeführt werden, bis die Anwendung beendet wurde. In der Zeile 97 wird mit Hilfe der Methode „takeScreenShot()“, welche durch die Robot-Klasse ein unkomprimiertes Bild liefert, das Bildschirmfoto erstellt.

Dieses Bild wird in den Zeilen 102 bis 109 durch den javaeigenen JPEG-Encoder in ein komprimiertes Bild des Typs „jpeg“ encodiert und zuvor in einen Datenstrom geladen.

In der Zeile 112 versucht die Methode „run()“ auf den Datenstrom des verbundenen Klienten zuzugreifen und das encodierte Bild in Form von Byte-Daten an die Gegenstelle zu übertragen. Die Funktion „sleep(250)“ in Zeile 119 definiert die Zeitspanne in Millisekunden, wie lange die gesamten Anweisungen in der While-Schleife nach einer jeden Ausführung pausieren müssen.

5.6.3 Forschungsbedingte Ansätze zur Realisierung von NAT-Überwindung und direkter Kommunikation über das Internet

Die Aufgabe dieses Teilkapitels besteht darin, zwei Ansätze darzustellen, welche während der Forschung zur Realisierung direkter Kommunikation zweier Systeme über das Internet, unter dem Versuch NAT zu überwinden, entstanden sind.

Die erste Beispielanwendung läuft in einer Client-Server Umgebung, die mit Hilfe von zwei Rechnern umgesetzt wurde und beruht auf dem verbindungsorientierten Protokoll TCP. Für die Realisierung dieser Applikation wurde ein DynDNS-Account erstellt und dessen Ziel-Adresse auf die öffentliche IP-Adresse eines Routers geleitet. Der erste Versuch, eine Verbindung zu dem Server ohne Portforwarding im Router aufzubauen, ist gescheitert, da die NAT-Funktionalität alle ankommenden Datenpakete geblockt hat. Wurde der Zugriff auf die Ports, durch manuelles Portforwarding, erlaubt, konnte die Verbindung problemlos hergestellt werden. Aus diesem Grund ist diese Applikation ein nennenswertes Beispiel, wie ein Java-Server im Internet und die zugehörige Client-Anwendung in einem privaten Netzwerk aussehen könnte. Die folgende Abbildung 37 zeigt den TCP-Server.

```
17 public class ChatServer {
18
19     private static void handleClient(Socket client) throws IOException {
20         System.out.println("Client connected...");
21         Scanner input = new Scanner(client.getInputStream());
22         PrintWriter output = new PrintWriter(client.getOutputStream(), true);
23         output.println("Hallo " + input.nextLine());
24         output.flush();
25     }
26
27     public static void main (String[] args) throws IOException {
28         ServerSocket server = new ServerSocket(4444);
29
30         while(true) {
31             Socket client = null;
32             try {
33                 client = server.accept();
34                 handleClient(client);
35             }
36             catch (IOException ex) {
37                 ex.printStackTrace();
38
39             }
40             finally {
41                 if(client != null) {
42                     try {
43                         client.close();
44
45                     }
46                     catch (IOException ex) {
47                         ex.printStackTrace();
48                     }
49                 }
50             }
51         }
52     }
53
54 }
```

Abbildung 37: Java Code – TCP-Server

In der Zeile 28 des oben dargestellten Quellcodes wird das ServerSocket-Objekt mit dem Port 4444, welcher im Router für die Verwendung eingerichtet wurde, initialisiert. Im weiteren Verlauf des Programms hat der Server nur noch die Aufgabe eingehende Verbindungen von Teilnehmern zu bearbeiten und diese mit einer kurzen Nachricht zu begrüßen. In Zeile 33 wird die Methode „accept()“ des Socket-Objektes aufgerufen, welche dafür zuständig ist, um auf eingehende Client-Requests zu warten. Sobald eine Verbindung eines Klienten eingeht, springt das Programm in die nächste Zeile und ruft die Methode „handleClient()“ auf. Diese Methode liest in Zeile 21 den sogenannten Eingangs-Datenstrom des Clients aus und schickt zugleich die Antwort „Hallo“ zuzüglich den erhaltenen Daten an den Client zurück (Zeilen 22, 23 und 24).

Die Methode „flush()“ dient dazu, die restlichen Daten aus dem Puffer des Datenstroms zu schreiben. Die folgende Abbildung stellt die Methode „connectToServer()“ der Client-Applikation vor.

```
130     private void connectToServer(String ip, int port) throws IOException {
131         Socket server = null;
132 
133         taStatus.setEnabled(true);
134         taStatus.setText("Verbindung zum Server wird aufgebaut... \n");
135 
136         Inet4Address inet = (Inet4Address) Inet4Address.getByName(ip);
137 
138         try {
139             server = new Socket(ip, port);
140             Scanner input = new Scanner(server.getInputStream());
141             PrintWriter output = new PrintWriter(server.getOutputStream(), true);
142             output.println(tfUserName.getText());
143             taStatus.append("Antwort vom Server: " + input.nextLine());
144         }
145         catch (UnknownHostException ex) {
146             ex.printStackTrace();
147         }
148         catch (IOException ex) {
149             ex.printStackTrace();
150         }
151         finally {
152             if(server != null) {
153                 try {
154                     server.close();
155                 }
156                 catch (IOException ex) {
157                     ex.printStackTrace();
158                     System.exit(-1);
159                 }
160             }
161         }
162     }
```

Abbildung 38: Java Code – TCPClient

In Zeile 139 des oben gezeigten Quellcodes wird die Verbindung zum Server hergestellt. Wie bei dem Server, werden auch hier die Ein- und Ausgabendatenströme der Gegenstelle geöffnet um Nachrichten auszutauschen. In diesem Fall wird der Inhalt des Textfeldes „tfUserName“ an den Server übertragen. Trägt man beispielsweise „Ringo“ in dieses Textfeld ein, würde man „Hallo Ringo“ als Antwort vom Server erhalten. Der Aufruf der Methode „connectToServer()“ wird in der folgenden Abbildung dargestellt.

```
118         try {
119         connectToServer("yato2007lskiri.dyndns.org", 4444);
120         }
121         catch (IOException ex) {
122             ex.printStackTrace();
123         }
```

Abbildung 39: Java Code - Aufruf connectToServer TCP

Der in Abbildung 39 dargestellte Aufruf der Methode zeigt, dass der Server über die angelegte DynDNS-Domain, welche auf die öffentliche IP-Adresse eines Routers weiterleitet, erreichbar ist. Da der Port 4444 im Router manuell eingerichtet wurde, kann eine Kommunikation und damit ein erfolgreicher Datenaustausch zwischen Client und Server stattfinden.

Im weiteren Teil dieses Kapitels wird eine UDP-Chat-Applikation vorgestellt, mit welcher es unter vier verschiedenen Testumgebungen zu drei Erfolgserlebnissen gekommen ist. Demzufolge ist es mit diesem Programm ansatzweise gelungen Datenpakete zwischen zwei NAT-gestützten Endpunkten, ohne Portforwarding, auszutauschen. Die Idee hinter diesem Ansatz liegt darin, beide Parteien parallel eine Art Bekanntmachung in Form eines Acknowledge-Strings senden zu lassen, während sie zeitgleich auf dem identischen Port auch bereit sind Daten zu empfangen. Der folgende Quellcode-Auszug zeigt den Konstruktor und Methode „ack()“ der Klasse „UDPSender“. Dabei wird verdeutlicht, wie der Acknowledge-String an die jeweilige Gegenstelle gesendet wird.

```
26      public UDPSender(DatagramSocket socket, String ip, String text) {
27          this.ip = ip;
28          this.clientSocket = socket;
29          this.data = text;
30          this.address = new InetSocketAddress(ip, 9890);
31          ack();
32      }
33
34      private void ack() {
35          byte[] ackData = new byte[1024];
36          String ackString = "abcdefg";
37
38          while (ChatGui.notify == false) {
39              try {
40                  ackData = ackString.getBytes("UTF-8");
41                  DatagramPacket ackPacket =
42                      new DatagramPacket(ackData, ackData.length, address);
43
44                  clientSocket.send(ackPacket);
45                  System.out.println("Ack gesendet...");
46                  try {
47                      Thread.sleep(1000);
48                  }
49                  catch (InterruptedException iEx) {
50
51                  }
52              }
53              catch (IOException ex) {
54                  System.out.println(ex);
55              }
56          }
57      }
```

Abbildung 40: Java Code – UDPSender

Die Klasse „UDPSender“ ist thread-fähig und wird auch als solche behandelt. Das heißt die Methode ack() läuft solange in einem 1-Sekunden-Rhythmus weiter, bis die Boolean-Variable „notify“, durch die parallellaufende Thread-Klasse „UDPReciever“, nach Erhalt des Acknowledge-Strings der Gegenstelle, auf den Wert „true“ gesetzt wird. Nachdem die Bekanntmachung stattgefunden hat und die „notify“-Variable den Wert „true“ besitzt kann in Folge dessen, die „run()“-Methode des Threads gestartet werden, welche den eigentlichen Chat ausmacht.

```
59      public void run() {
60          while (ChatGui.messageIsSent == false) {
61              try {
62                  byte[] sendData = new byte[1024];
63                  sendData = data.getBytes("UTF-8");
64                  DatagramPacket sendPacket =
65                      new DatagramPacket(sendData, sendData.length, address);
66
67                  clientSocket.send(sendPacket);
68                  String buffer = new String(sendData, "UTF-8");
69                  ChatGui.taStatus.append("\n Du: " + buffer);
70                  ChatGui.messageIsSent = true;
71              }
72              catch (Exception ex) {
73                  ex.printStackTrace();
74              }
75              finally {
76                  this.interrupt();
77              }
78          }
79      }
```

Abbildung 41: Java Code - UDPSender run

Die hier dargestellte Methode wird jedes Mal ausgeführt, sobald der Anwender die Nachricht abschicken möchte. Die zu sendende Message empfängt die Klasse erneut durch den Konstruktor und wird in der Zeile 63 als Bytewert in ein Byte-Array geschrieben. Die Methode „clientSocket.send()“ sendet das zuvor definierte Datagram-Paket an den weiteren Teilnehmer. Im Anschluss wird die Boolean-Variable „messageIsSent“ auf den Wert „true“ gesetzt, damit die Zeile 60 definierte While-Schleife vorzeitig verlassen werden kann. Mit dem Verlassen der While-Schleife, ist auch der Thread beendet und die Nachricht wurde versandt. Während der Testzeit mit dieser Applikation, haben sich die Nachteile des UDP-Protokolls deutlich bemerkbar gemacht. Durch die verbindungslose Übertragung der Datenpakete, sind Daten verloren gegangen. Bei dem Senden von 10 Nachrichten an eine Gegenstelle, kam es zu einer erfolgreichen Durchsatzrate von 90 Prozent.

6 Fazit

Zu dieser Arbeit lassen sich abschließend verschiedene persönliche Eindrücke festhalten. Der Umstieg von der Programmiersprache C-Sharp auf Java war anfangs gewöhnungsbedürftig. Trotz der ähnlichen Syntax und allgemeinen Vorgehensweise in der Programmgestaltung, fühlte sich Java zum einen massiv langsamer und zum anderen komplizierter an. Nach den anfänglichen Erfolgserlebnissen in der Netzwerkprogrammierung, ergaben sich bald darauf die ersten Probleme. Abgesehen von der mangelnden Nutzerverwaltung in der Datenbank und die Sorge über derer Sicherheit bis hin zu dem fehlenden javafähigen Webserver. Auch wenn zwischen den funktionalen Anforderungen und der tatsächlichen Realisierung eine größere Differenz besteht als zu Beginn erwartet wurde, ziehe ich eine äußerst positive Bilanz. Durch die vielen Testapplikationen im Bezug auf die Netzwerkprogrammierung und Datenbanksicherheit, konnte ich mein Fachwissen in diesen Bereichen deutlich erweitern. Der Wechsel zu der Programmiersprache Java, stellt sich im Nachhinein als kein Nachteil ein, sondern eröffnet mir eine völlig andere Sichtweise auf die Plattformunabhängigkeit. Ich kann mir durchaus vorstellen weiterhin mit Java Applikationen zu entwickeln und vor allem das begonnene Projekt HBOSS unter besseren Bedingungen zu erweitern.

Ich bin der Meinung, dass man diesem Projekt noch etwas Zeit schenken sollte, bevor es veröffentlicht wird. Die grundlegende Funktionalität ist zwar durch das Ticketsystem gegeben, doch wäre die Aufmerksamkeit der Anwender wesentlich höher, wenn diese Applikation, den Schritt an die Öffentlichkeit, in vollem Umfang wagen würde.

Abbildungsverzeichnis

Tabellenverzeichnis

Anlage A

```php
function sendMail($userEmail) {
    $time = time();
    $reciever = $userEmail;
    $subject = "Haendlerbund Online Support System";

    //attachement-file
    $attFile = "hboss_client.zip";
    $attFileName = "hboss_client.zip";

    //setting up the headerinformations
    $header = "From: Haendlerbund <info@haendlerbund.de>";

    $splitter = md5(uniqid(time()));

    $header .= "\n";
    $header .= "MIME-Version: 1.0";
    $header .= "\n";
    $header .= "Content-Type: multipart/mixed;
                boundary=$splitter";
    $header .= "\n\n";
    $header .= "This is a multipart message in MIME format";
    $header .= "\n";
    $header .= "--$splitter";
    $header .= "\n";
    $header .= "Content-Type: text/plain";
    $header .= "\n";
    $header .= "Content-Transfer-Encoding: 8bit";
    $header .= "\n\n";
    $header .= "Haendlerbund Online Support System";
    $header .= "\n";
    $header .= "--$splitter";
    $header .= "\n";
    $header .= "Content-Type: application/zip;
                name=$attFileName";
    $header .= "\n";
    $header .= "Content-Transfer-Encoding: base64";
    $header .= "\n";
    $header .= "Content-Disposition: attachment;
                filename=$attFilename";
    $header .= "\n\n";

    $fileContent = fread(fopen($attFile, "r"), filesize($attFile));

    $header .= chunk_split(base64_encode($fileContent));
    $header .= "\n";
    $header .= "--$splitter--";

    //sending the mail
    mail($reciever, $subject, "", $header);

    //waiting for execution
    sleep(2);
}
```

Abbildung 42: PHP Code - Mailheader für Dateianhänge

Literaturverzeichnis

Becker, R., Liebsch, F., Michel, Y. R., & Müller, D. (2004). *JXTA: Einführung und Überblick.* FU Berlin. Berlin: FU Berlin.

E-Soft Inc. (2010 йил 01-05). *Apache Module Report*. Retrieved 2011 йил 05-03 from http://www.securityspace.com/s_survey/data/man.201004/apachemods.html

E-Soft Inc. (2010 йил 01-05). *Server Details*. Retrieved 2011 йил 05-03 from http://www.securityspace.com/s_survey/sdata/201004/servers.html

E-Soft Inc. (2010 йил 01-05). *Server Market Share*. Retrieved 2011 йил 05-03 from http://www.securityspace.com/s_survey/sdata/201004/index.html

Franzis. (2004). *PHP 5. espresso.* : Franzis.

Händlerbund. (2011 йил 01-03). *Händlerbund*. Retrieved 2011 йил 01-03 from http://www.haendlerbund.de

Höhne, T. (29. 03 2011). Re: Technische Frage zur Sicherheit der MySQL-Datenbank. Friedersdorf, Sachsen, Deutschland: Neue Medien Münnich.

Kredel, H. (2002-2007). *JXTA.* Universität Mannheim, Rechenzentrum. Mannheim: Universität Mannheim.

Michler, M. (2011). *Auflistung der Supportanfragen.* Leipzig: Händlerbund.

Microsoft. (2011 йил -). *Microsoft Technet*. Retrieved 2011 йил 08-03 from Microsoft Technet: http://technet.microsoft.com/de-de/library/cc785220(WS.10).aspx

Morgan Kaufmann Publishers. (2004). *TCP/IP Sockets in C#.* San Francisco: Elsevier.

MySQL, Oracle. (1997-2010 йил -). *MySQL*. Retrieved 2011 йил 07-03 from MySQL: http://dev.mysql.com/doc/refman/5.1/de/connector-j-usagenotes-basic.html#connector-j-examples-connection-drivermanager

Reid, F. (2004). *Network Programming in .NET.* Oxford: Elsevier Digital Press.

Sams Puplishing. (2002). *JXTA: Java P2P Programming.* USA.

The PHP Group. (2001-2011). *PHP.net*. Retrieved 2011 йил 04-03 from http://www.php.net

Ullenboom, C. (2009). *Java ist auch eine Insel.* Galileo Computing.

Wikipedia. (2011). *Wikipedia, JXTA*. (Wikimedia Foundation Inc.) Retrieved 2011 йил 02-04 from http://de.wikipedia.org/wiki/JXTA

ZDNET. (2007 йил 27-09). *ZDNET*. Retrieved 2011 йил 04-05 from Code-Dieben das Handwerk legen: http://www.zdnet.de/anwendungsentwicklung_obfuskation_code_dieben_das_handwerk_legen_story-20000201-39157887-1.htm

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Leipzig, 31.05.2011

Ringo Lewandrowski